# Eigen4AutoIt

# Table of contents


Eigen4AutoIt ..... 11
Introduction ..... 12
Why Bother? ..... 14
License ..... 15
Installation ..... 19
Credits ..... 22
History / ChangeLog ..... 23
Tutorials ..... 29
Basics ..... 29
Slicing and Dicing ..... 39
Regression ..... 45
Principal Components Analysis ..... 57
Pooled Multi-Processing ..... 69
Eigen4AutoIt Function Reference ..... 83
Function Notes ..... 98
Complex Function Notes ..... 100
Work Environment ..... 102
CleanUp ..... 105
Debugmode_Off ..... 106
Debugmode_On ..... 107
Hide_Errors ..... 108
Hide_Warnings ..... 109
IsComplex ..... 110
ReleaseFromIndex ..... 111
ReleaseFromIndex_Except ..... 112
ReleaseFromIndex_ToIndex ..... 113
ReleaseFromIndex_ToIndex_Except ..... 115
ReleaseFromMarker ..... 116
ReleaseFromMarker_Except ..... 117
ResetListMarker ..... 118
SetListMarker ..... 120
SetMatrixType ..... 121
Show_AllFunctions ..... 122
Show_Constants_Nist ..... 123
Show_Constants_Sykora1 ..... 124
Show_Constants_Sykora2 ..... 125
Show_DllFunctions ..... 126
Show_EnvironmentVars ..... 127
Show_Errors ..... 128
Show_MatrixList ..... 129
Show_Warnings ..... 130
StartUp ..... 131
Test_EigenAssert ..... 133
Matrix Management ..... 134
Matrix Creation ..... 135
CreateMatrix ..... 135
CreateMatrix_Constant ..... 137

CreateMatrix_FromA ..... 138
CreateMatrix_FromA_Transposed ..... 139
CreateMatrix_FromABcols ..... 140
CreateMatrix_FromABcols_Transposed ..... 142
CreateMatrix_FromAblock ..... 143
CreateMatrix_FromAblock_Transposed ..... 144
CreateMatrix_FromABrows ..... 145
CreateMatrix_FromABrows_Transposed ..... 146
CreateMatrix_FromAcols ..... 148
CreateMatrix_FromAcols_Tranposed ..... 149
CreateMatrix_FromArows ..... 150
CreateMatrix_FromArows_Transposed ..... 152
CreateMatrix_Identity ..... 153
CreateMatrix_LinSpaced_ColMajor ..... 154
CreateMatrix_LinSpaced_RowMajor ..... 156
CreateMatrix_Ones ..... 157
CreateMatrix_Random ..... 158
CreateMatrix_Zero ..... 159
CreateOutPut_FromDims ..... 160
CreateOutPut_FromIDs ..... 162
Add_ExistingMatrix ..... 163
CloneArray ..... 164
CloneMatrix ..... 165
CreateMatrix_FromArray ..... 166
CreateArray_FromMatrix ..... 168
GetActiveMatrix ..... 169
GetMatrixCols ..... 170
GetMatrixRows ..... 171
GetMatrixType ..... 172
GetMatrixTypeID ..... 173
IsActiveMatrix ..... 174
_MatrixDisplay ..... 175
ReadMatrixValue ..... 178
RedefineMatrix ..... 179
Redim_ExistingMatrix ..... 180
ReleaseMatrix ..... 181
ReleaseMatrix_All ..... 183
ReleaseMatrix_All_Except ..... 184
ResetActiveMatrix ..... 185
ResetActiveMatrix_Single ..... 186
SetActiveMatrix ..... 187
SplitMatrix_FromAcol ..... 189
SplitMatrix_FromAcol_Transposed ..... 190
SplitMatrix_FromArow ..... 191
SplitMatrix_FromArow_Transposed ..... 193
WriteMatrixValue ..... 194
File I/O and Type Conversion ..... 196
ConvertMatrix_ToComplexDouble ..... 198
ConvertMatrix_ToComplexFloat ..... 199
ConvertMatrix_ToDouble ..... 201

ConvertMatrix_ToFloat ..... 202
ConvertMatrix_ToInt ..... 203
ConvertMatrixFile_ToComplexDouble ..... 205
ConvertMatrixFile_ToComplexFloat ..... 206
ConvertMatrixFile_ToDouble ..... 208
ConvertMatrixFile_ToFloat ..... 209
ConvertMatrixFile_ToInt ..... 211
LoadMatrix ..... 212
Redim_ExistingMatrixFile ..... 213
SaveMatrix ..... 215
Set ..... 216
SetConstant ..... 217
SetConstant_Block ..... 218
SetConstant_Col ..... 219
SetConstant_Diag ..... 220
SetConstant_Row ..... 221
SetIdentity ..... 222
SetIdentity_Block ..... 223
SetLinSpaced_Col ..... 224
SetLinSpaced_ColMajor ..... 226
SetLinSpaced_Diag ..... 227
SetLinSpaced_Row ..... 228
SetLinSpaced_RowMajor ..... 229
SetOnes ..... 230
SetOnes_Block ..... 231
SetOnes_Col ..... 232
SetOnes_Diag ..... 233
SetOnes_Row ..... 233
SetRandom ..... 234
SetRandom_Block ..... 235
SetRandom_Col ..... 236
SetRandom_Diag ..... 237
SetRandom_Row ..... 238
SetZero ..... 239
SetZero_Block ..... 240
SetZero_Col ..... 241
SetZero_Diag ..... 242
SetZero_Row ..... 243
Copy ..... 244
Copy_A_ToB ..... 246
Copy_A_ToBimag ..... 247
Copy_A_ToBreal ..... 248
Copy_Ablock_ToAblock ..... 249
Copy_Ablock_ToBblock ..... 251
Copy_Acol_ToBcol ..... 252
Copy_Acol_ToBdiag ..... 253
Copy_Acol_ToBrow ..... 254
Copy_Adiag_ToBcol ..... 255
Copy_Adiag_ToBdiag ..... 257
Copy_Adiag_ToBrow ..... 258

Copy_Adiag_ToBvector ..... 259
Copy_Aimag_ToB ..... 260
Copy_Aimag_ToBimag ..... 261
Copy_Aimag_ToBreal ..... 262
Copy_ALower_ToBLower ..... 263
Copy_ALower_ToBUpper ..... 264
Copy_Arow_ToBcol ..... 265
Copy_Arow_ToBdiag ..... 266
Copy_Arow_ToBrow ..... 267
Copy_ArrayData_ToMatrix ..... 268
Copy_Areal_ToB ..... 269
Copy_Areal_ToBimag ..... 271
Copy_Areal_ToBreal ..... 272
Copy_AStrictlyLower_ToBLower ..... 273
Copy_AStrictlyUpper_ToBUpper ..... 274
Copy_AUnitLower_ToBLower ..... 275
Copy_AUnitUpper_ToBUpper ..... 276
Copy_AUpper_ToBLower ..... 277
Copy_AUpper_ToBUpper ..... 278
Copy_Avector_ToBdiag ..... 279
Copy_MatrixData_ToArray ..... 280
Swap ..... 281
Swap_Ablock_Ablock ..... 282
Swap_Ablock_Bblock ..... 283
Swap_Acol_Acol ..... 285
Swap_Acol_Adiag ..... 286
Swap_Acol_Arow ..... 287
Swap_Acol_Bcol ..... 287
Swap_Acol_Bdiag ..... 289
Swap_Acol_Brow ..... 290
Swap_Adiag_Bdiag ..... 291
Swap_ALower_BLower ..... 292
Swap_Areal_Aimag ..... 293
Swap_Areal_Bimag ..... 294
Swap_Arow_Adiag ..... 295
Swap_Arow_Arow ..... 296
Swap_Arow_Bdiag ..... 297
Swap_Arow_Brow ..... 298
Swap_AStrictlyLower_BStrictlyLower ..... 299
Swap_AStrictlyUpper_BStrictlyLower ..... 300
Swap_AStrictlyUpper_BStrictlyUpper ..... 301
Swap_AUpper_ALower ..... 303
Swap_AUpper_BLower ..... 303
Swap_AUpper_BUpper ..... 304
Cellwise Operators ..... 306
Binary Operators ..... 308
CwiseBinaryOp ..... 309
CwiseBinaryOp_Block ..... 311
CwiseBinaryOp_Block_InPlace ..... 312
CwiseBinaryOp_Col ..... 314

CwiseBinaryOp_Col_InPlace ..... 316
CwiseBinaryOp_ColCol ..... 317
CwiseBinaryOp_ColCol_InPlace ..... 319
CwiseBinaryOp_ColRow ..... 321
CwiseBinaryOp_ColRow_InPlace ..... 322
CwiseBinaryOp_Colwise ..... 324
CwiseBinaryOp_Colwise_InPlace ..... 326
CwiseBinaryOp_ColwiseCol ..... 327
CwiseBinaryOp_ColwiseCol_InPlace ..... 329
CwiseBinaryOp_InPlace ..... 330
CwiseBinaryOp_Row ..... 332
CwiseBinaryOp_Row_InPlace ..... 333
CwiseBinaryOp_RowCol ..... 335
CwiseBinaryOp_RowCol_InPlace ..... 336
CwiseBinaryOp_RowRow ..... 338
CwiseBinaryOp_RowRow_InPlace ..... 340
CwiseBinaryOp_Rowwise ..... 342
CwiseBinaryOp_Rowwise_InPlace ..... 343
CwiseBinaryOp_RowwiseRow ..... 345
CwiseBinaryOp_RowwiseRow_InPlace ..... 346
Show_BinaryOperators ..... 348
Conditional Operators ..... 349
ConditAll ..... 349
ConditAll_Col ..... 351
ConditAll_Row ..... 352
ConditAny ..... 354
ConditAny_Col ..... 355
ConditAny_Row ..... 357
ConditBinaryOp ..... 358
ConditBinaryOp_InPlace ..... 360
ConditCount ..... 362
ConditCount_Col ..... 363
ConditCount_Row ..... 365
ConditScalarOp ..... 366
ConditScalarOp_InPlace ..... 368
ConditUnaryOp ..... 370
ConditUnaryOp_InPlace ..... 372
Show_ConditOperators ..... 374
Scalar Operators ..... 375
CwiseScalarOp ..... 376
CwiseScalarOp_Block ..... 377
CwiseScalarOp_Block_InPlace ..... 379
CwiseScalarOp_Col ..... 381
CwiseScalarOp_Col_InPlace ..... 382
CwiseScalarOp_ColCol ..... 384
CwiseScalarOp_ColCol_InPlace ..... 385
CwiseScalarOp_ColRow ..... 387
CwiseScalarOp_ColRow_InPlace ..... 388
CwiseScalarOp_InPlace ..... 390
CwiseScalarOp_Row ..... 391

CwiseScalarOp_Row_InPlace ..... 393
CwiseScalarOp_RowCol ..... 394
CwiseScalarOp_RowCol_InPlace ..... 396
CwiseScalarOp_RowRow ..... 398
CwiseScalarOp_RowRow_InPlace ..... 399
Show_ScalarOperators ..... 401
Unary Operators ..... 402
CwiseUnaryOp ..... 403
CwiseUnaryOp_Block ..... 404
CwiseUnaryOp_Block_InPlace ..... 406
CwiseUnaryOp_Col ..... 408
CwiseUnaryOp_Col_InPlace ..... 409
CwiseUnaryOp_ColCol ..... 411
CwiseUnaryOp_ColCol_InPlace ..... 413
CwiseUnaryOp_ColRow ..... 414
CwiseUnaryOp_ColRow_InPlace ..... 416
CwiseUnaryOp_InPlace ..... 418
CwiseUnaryOp_Row ..... 419
CwiseUnaryOp_Row_InPlace ..... 421
CwiseUnaryOp_RowCol ..... 422
CwiseUnaryOp_RowCol_InPlace ..... 424
CwiseUnaryOp_RowRow ..... 426
CwiseUnaryOp_RowRow_InPlace ..... 427
Show_UnaryOperators ..... 429
Reduction ..... 430
IsCwiseEqual_AB ..... 431
IsCwiseEqual_ToConstant ..... 433
MatrixSpecs ..... 434
MatrixSpecs_Block ..... 435
MatrixSpecs_Block_Single ..... 436
MatrixSpecs_Col ..... 437
MatrixSpecs_Col_Single ..... 438
MatrixSpecs_Colwise_Single ..... 440
MatrixSpecs_Diag ..... 441
MatrixSpecs_Diag_Single ..... 442
MatrixSpecs_Row ..... 443
MatrixSpecs_Row_Single ..... 444
MatrixSpecs_Rowwise_Single ..... 445
MatrixSpecs_Single ..... 447
Show_MatrixSpecs ..... 448
Show_MatrixSpecs_Current ..... 449
Transformation ..... 450
Adjoint ..... 451
Adjoint_InPlace ..... 452
Inverse ..... 453
Inverse_InPlace ..... 454
PseudoInverse ..... 455
Reverse ..... 456
Reverse_Block ..... 457
Reverse_Block_InPlace ..... 458

Reverse_Col ..... 460
Reverse_Col_InPlace ..... 461
Reverse_Diag ..... 462
Reverse_Diag_InPlace ..... 463
Reverse_InPlace ..... 464
Reverse_Row ..... 465
Reverse_Row_InPlace ..... 466
Transpose ..... 467
Transpose_InPlace ..... 468
Multiplication ..... 469
CrossProduct ..... 470
DotProduct ..... 471
Multiply_A_BdivC ..... 472
Multiply_A_BminusC ..... 474
Multiply_A_BplusC ..... 475
Multiply_AA ..... 476
Multiply_AA_InPlace ..... 477
Multiply_AAt ..... 478
Multiply_AAt_InPlace ..... 479
Multiply_AB ..... 480
Multiply_AB_divC ..... 481
Multiply_AB_InPlace ..... 482
Multiply_AB_minusC ..... 483
Multiply_AB_plusC ..... 485
Multiply_ABC ..... 486
Multiply_ABC_InPlace ..... 487
Multiply_ABCD ..... 488
Multiply_ABCD_InPlace ..... 490
Multiply_ABCDE ..... 491
Multiply_ABCDE_InPlace ..... 493
Multiply_ABt ..... 494
Multiply_ABt_InPlace ..... 495
Multiply_AtA ..... 496
Multiply_AtA_InPlace ..... 497
Multiply_AtAt ..... 498
Multiply_AtAt_InPlace ..... 499
Multiply_AtB ..... 500
Multiply_AtB_InPlace ..... 502
Multiply_AtBt ..... 503
Multiply_AtBt_InPlace ..... 504
OuterProduct ..... 505
Decomposition ..... 506
Decomposition Outputs: Matrices ..... 507
Decomposition Outputs: Dimensions ..... 509
Decomp_Choleski ..... 510
Decomp_ComplexSchur ..... 512
Decomp_Hessenberg ..... 513
Decomp_HouseholderQR ..... 515
Decomp_JacobiSVD ..... 518
Decomp_LowerUpper ..... 520

Decomp_RealQZ ..... 522
Decomp_RealSchur ..... 524
Decomp_Tridiagonalization ..... 526
Show_DecompSpecs ..... 528
Show_DecompSpecs_Current ..... 529
EigenSolvers ..... 529
EigenSolver ..... 532
EigenSolver_Complex ..... 534
EigenSolver_Generalized ..... 535
EigenSolver_Generalized_SelfAdjoint ..... 537
EigenSolver_SelfAdjoint ..... 539
Statistics ..... 541
Center ..... 542
Center_Colwise ..... 543
Center_Colwise_InPlace ..... 544
Center_InPlace ..... 545
Center_Rowwise ..... 546
Center_Rowwise_InPlace ..... 547
Correlation_AB ..... 548
Correlation_ColCol ..... 550
Correlation_RowRow ..... 551
Covariance_Colwise ..... 552
Covariance_Rowwise ..... 553
Kurtosis ..... 554
Kurtosis_Col ..... 555
Kurtosis_Row ..... 556
Normalise ..... 558
Normalise_Colwise ..... 559
Normalise_Colwise_InPlace ..... 560
Normalise_InPlace ..... 561
Normalise_Rowwise ..... 562
Normalise_Rowwise_InPlace ..... 563
Skewness ..... 564
Skewness_Col ..... 565
Skewness_Row ..... 566
StDev ..... 567
StDev_Col ..... 568
StDev_Colwise ..... 570
StDev_Row ..... 571
StDev_Rowwise ..... 572
Advanced ..... 573
LeastSquares ..... 574
Misfit ..... 576
Misfit_Col ..... 577
Misfit_Colwise ..... 579
PartialLeastSquares ..... 581
PCA ..... 583
PCA_ReDim ..... 585
RelativeError ..... 587
RelativeError_Col ..... 589

RelativeError_Colwise ..... 590
RsqAdjusted ..... 592
Rsquared ..... 594
Rsquared_Col ..... 596
Rsquared_Colwise ..... 598
SSR ..... 600
SSR_Col ..... 601
SSR_Colwise ..... 603
Appendix ..... 605
Eigen4AutoIt Globals ..... 605
Eigen4AutoIt Limitations ..... 607
Eigen4AutoIt Error Codes ..... 608

# Eigen4AutoIt

version **2.5**



Download the latest version  Eigen Homepage  AutoIt Homepage

## Features

- free, **fast** matrix computing environment for Windows (runs under Wine on Linux too)
- built upon the continually expanding, open-source **Eigen** codebase (stable release **3.2.7**)
- integer, single and double precision, real and complex matrices
- over three hundred native functions (AutoIt source code included)
- over two hundred **Eigen** calls (C++ source code included)
- **interpreter** (direct execution) *and* **compiler** (stand-alone executables (with external dll))
- animated **Function Selector** utility provides quick access to all function templates
- Debugging Mode integrates Eigen's own assert error messages for rapid development and testing
- Computing Mode supports Streaming SIMD Extensions 2 (SSE2) architecture
- anmated 3D visualisations with the **MatrixViewer** utility (with separate MatrixList viewer)
- support for multi-processing on networked clusters using the **Pool** environment
- exchange data easily between native binary format and ASCII, Excel, and Xbase files, or **AutoIt** arrays
- extensive documentation, online Help, call stack tracing, meaningful error messages
- Tutorials with scripts, plus test scripts for each function section
- over seven hundred alias wrappers for flexibility and ease-of-use
- full access to **AutoIt**'s powerful features (GUIs, WinAPI, COM, ActiveX, GDI+, SQL, IE, Excel, Word, external dlls)

# Table of Contents


- Introduction
- Why Bother?
- License
- Installation
- Credits
- History / ChangeLog

## Tutorials

- Basics
- Slicing and Dicing
- Regression (fitting a function to data)
- Principal Components Analysis (PCA)
- Pooled Multi-Processing

## Function Reference

- Eigen4AutoIt Function Reference
- Work Environment
- Matrix Management
- File I/O and Type Conversion
- Set
- Copy
- Swap
- Cellwise Operators
- Reduction
- Transformation
- Multiplication
- Decomposition
- EigenSolvers
- Statistics

## Appendix

- Eigen4AutoIt Globals
- Eigen4AutoIt Limitations
- Eigen4AutoIt Error Codes



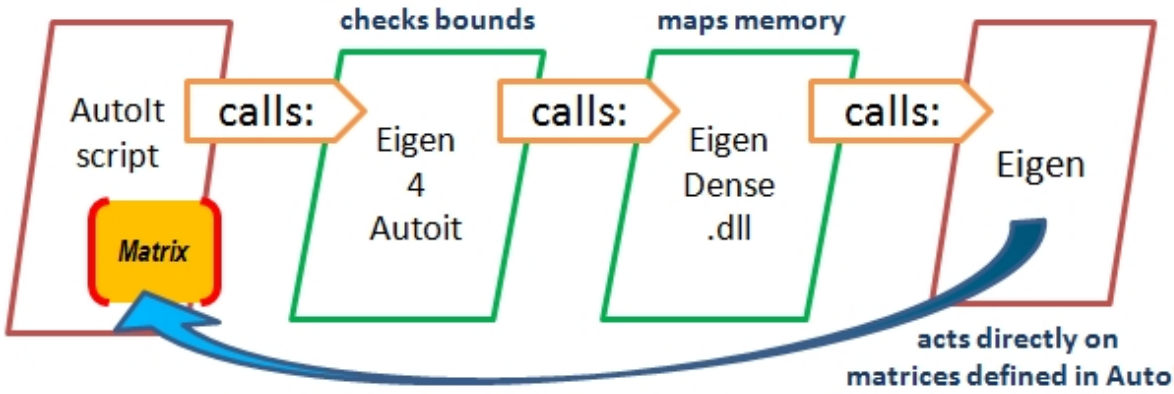

## Introduction

# Introduction

The Eigen C++ template library is a great environment for scientific computing; it is fast, reliable, extensive, and reasonably well-documented (here). It is also 100% free, and does not rely on any external dependencies. However, it does have two mutually related drawbacks in its name: "C++" and "template library." C++ is strong-typed, autistically user-*un*friendly, and an almost unsurmountable hurdle for any beginner. The term "template library" (admittedly a feature when using C++) implies that any functions you call are only instantiated upon compilation; no internal library exists that other languages can hook into.

AutoIt is a dynamic scripting language that is incredibly easy to learn and understand. It is exceptionally flexible in interpreting code, and simple functions can access the huge resources of the entire Windows operating system environment, including graphics, GUIs, COM, ActiveX, and so forth. It also has a large international user base, with unusually friendly and helpful forums in a number of languages. The AutoIt environment is, however, weak at scientific computation; array storage is fragmented in memory, type-based precision control is absent, and even when "compiled," codes run far slower than in C++.

The solution would be to simply merge the two environments, which is technically daunting, to say the least. The next best thing is to make Eigen's most important functionality directly accessible from within AutoIt scripts. To do this, both sides need a bridging interface that spans half of the chasm separating the two. Eigen requires dynamic link libraries that contain generic C++ wrappers for calling Eigen functions, either individually or in bulk, as complete, contained Eigen programmes with multiple outputs and parameter-based decision trees. AutoIt needs its own AutoIt wrappers to engage with these dlls, plus, it has to be able to handle Eigen Matrix variables directly, as these will be created and handled inside AutoIt scripts, while Eigen functions only manipulate their content.




The result is ***Eigen4AutoIt***. It consists of an AutoIt library of wrapper functions that contain extensive checks for specific bounds, dimensions, and other conditions, as well as providing matrix management, file I/O, type conversion, precision and debugging control, and two-way data exchange with files (internal: binary, external: ASCII delimited, Excel, and Xbase) and native AutoIt arrays. Actual Eigen calls are routed to the appropriate dll, where matrices created in AutoIt are memory-mapped for Eigen's powerful core functions to work on. All of this happens in-place, that is, no matrix *content* is ever transferred, only memory pointers. This means that apart from the negligible overhead of calling the dlls, computations in AutoIt are as fast as in Eigen's native environment, with the added benefit of not having to be compiled first. The main disadvantage is that you cannot construct ad-hoc expression templates that Eigen can optimise internally upon compilation.



## Why Bother?

# Why Bother?

This starter page is for people without a background in maths or science, who might be wondering what the big deal is. You may have heard of "matrix computing," or lured by promises of "powerful tools," "superfast algorithms," or cool shades, leather outfits, and "guns, lots of guns." Movie references aside, what benefits could **Eigen4AutoIt** have for you? What makes it so powerful, compared to the ordinary tools a computer-literate person has access to? Why would you possibly want to acquaint yourself with this daunting environment of over three hundred functions, and gibberish like "reduced dimensionality," linear solvers," and "cellwise operators"? Is it worth it?

Have you ever seen those ancient arcade games like Pac-Man or Space Invaders? Those blocky 2D graphics in primary colours? Now compare that in your mind with the lush 3D sandbox universes you can explore in modern gaming. Or imagine flip-book animations, containing a single, crudely drawn scene of a few hundred frames at most. Now think of today's feature-length CGI blockbusters with photo-real lighting and fleshed-out characters populating fully-realised worlds. Quite a difference, no? One of the major innovations that makes this stuff possible is **parallel computing**. Under the hood, both your new graphics card and Pixar's render farms are able to handle huge tasks, such as ray-tracing an entire scene with thousands of 3D objects, by dividing them into tiny chunks that are computed in parallel by an army of processing units.

The arithmetic from your school days is like Pac-Man and flipbook animations. It's kids' stuff. Matrix computing constitutes numerical parallelisation, and it kicks ass. Instead of animation frames, it handles matrices, replacing pixels with cells containing values. Just like the gamer or CGI animator who simply moves the camera viewpoint, affecting every pixel on screen, calling a single matrix function can affect millions of matrix cells simultaneously. And just as the screen size no longer matters to the software (just get a bigger graphics card or render farm!), it no longer matters how big your numerical problem is; whether you're dealing with two unknowns or two hundred at a time, ten equations or ten thousand. A matrix is a matrix is a matrix. That accounts for the "powerful tool" slogan. And it's got nothing to do with the hardware; you don't need a parallel computer or a GPU for this to work.

Whether you're building a predictive or descriptive model to fit your observations (see linear solvers), determining which of fifteen thousand features are truly important in recognising a human face (see PCA), or simply wishing to perform the same mathematical operation on a large number of inputs, matrices are your friend. Scientists love them because today's research produces terabytes of data (think genetic tests, the Large Hadron Collider, satellite data...) and matrices provide the best way to wield them as single entities, and extract useful, relevant information with a single function. So even though the hardware may internally still handle cells consecutively, it's the *conceptual* parallisation that makes formerly intractable problems so much easier to solve.

Of course, you have to play by some new rules. Matrix multiplication? Better make sure adjacent factors share a same-sized dimension. Store a transformation result back in your input? It's hip to be square, baby.  Matrix division? Sorry mate, doesn't exist. Solving an underdetermined system? Get ready to watch your precious result get sucked into null space and blow  up. There's much to learn in matrix land, and not all of it is pretty!

So is it worth it? Well, if you ever write code that deals with numerical data sets, or involves repetitive numerical tasks on many (or large) inputs, or quantifies unknowns, or needs to reduce huge piles of disparate information to a handful of essentials, then yes, absolutely. **Eigen4AutoIt** offers you the speed of Eigen's template library without the nuisance of C++, in an easy-going AutoIt environment where you don't need to compile each time you want to get things done. You can either call **Eigen4AutoIt**'s ready-made solutions that produce multiple outputs, or build your own optimised tools with a large set of simple, intuitively named, well-documented, tried and tested building blocks. Check out the code snippets in the Tutorials to get a flavour of how easy it is to get started. And when things get more complicated? Just remember the little internet start-up company that recognised that ranking web pages is actually a matrix eigenvalue problem. It did rather well. So go forth and matrix-multiply!



## License

# Software License

**Eigen4AutoIt**
**Author** : A.R.T. Jonkers <RTFC>
**WWW** : http://www.autoitscript.com/forum/files/file/319-eigen4autoit/
**Synopsis** : No warranties, no liability for damages, no support, no commercial usage without prior consent, no high-risk activities; free download, installation, use, and distribution of unlimited copies.

---

**END-USER LICENSE AGREEMENT FOR THIS SOFTWARE**
This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and the mentioned author of this Software for the software product identified above, which includes computer software and may include associated media, printed materials, and/or "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to all terms of this EULA, do not install or use the SOFTWARE PRODUCT.

**SOFTWARE PRODUCT LICENSE**
The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

**GRANT OF LICENSE**
Installation and Use. You may install and use an unlimited number of copies of the SOFTWARE PRODUCT.
Reproduction and Distribution. You may reproduce and distribute an unlimited number of copies of the SOFTWARE PRODUCT either in whole or in part; each copy should include all copyright and trademark notices, and shall be accompanied by a copy of this EULA. Copies of the SOFTWARE PRODUCT may be distributed as a standalone product or included with your own product.

Commercial Use. You may NOT use the SOFTWARE PRODUCT for commercial purposes. You may not sell, rent, lease, sub license, transfer, resell for profit or otherwise distribute the Software or any part thereof, without the Authors prior written consent.

Unauthorised use. You shall undertake any necessary steps to protect the Software against unauthorised use.

**COPYRIGHT**
All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by the Author of this Software, with the exception of freely available AutoIt and Eigen logo images.. The SOFTWARE PRODUCT is protected by copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material. You may not remove or obscure any copyright and trademark notices relating to the Software.
he author retains all rights, title and interest in and to the Software and all copies thereof, including copyrights, patents, trade secret rights, trademarks and other intellectual property rights. All rights not specifically granted in this Agreement, including International Copyrights, are reserved by the author.

**MISCELLANEOUS**
If you acquired this SOFTWARE PRODUCT in the European Union, this EULA is governed by the laws of the European Union. If this SOFTWARE PRODUCT was acquired outside the European Union, then local law may apply.

**NO WARRANTIES**

**THE SOFTWARE PRODUCT, PLUS ANY RELATED DOCUMENTATION AND ANY OTHER ASSOCIATED MATERIALS, ARE PROVIDED ON AN "AS IS" BASIS. THE AUTHOR MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTY OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. WITHOUT LIMITATION, YOU ASSUME SOLE RESPONSIBILITY FOR SELECTING THE SOFTWARE PRODUCT TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM THE SOFTWARE. THE AUTHOR MAKES NO WARRANTY THAT THE SOFTWARE WILL BE ERROR-FREE OR FREE FROM INTERRUPTIONS OR OTHER FAILURES. IN PARTICULAR, THE SOFTWARE IS NOT DESIGNED FOR USE IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE. THE AUTHOR EXPRESSLY DISCLAIMS ANY WARRANTY OF FITNESS FOR HIGH-RISK ACTIVITIES. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT REMAINS WITH YOU.**

The author warrants that he holds the proper rights allowing it to license the SOFTWARE PRODUCT and is not currently aware of any actions that may affect its rights to do so.

The author cannot guarantee that the SOFTWARE PRODUCT will work at all times and in all environments. For example, if you change your operating system, the SOFTWARE PRODUCT may not work anymore. You acknowledge and agree that such changes are fair and reasonable.

You should make sure that it is legal to use the SOFTWARE PRODUCT in your country or jurisdiction. The author only provides a license for You to use the software. It is Your responsibility is to make sure that You are allowed to use the SOFTWARE PRODUCT.

The author reserves the right at any time to cease any and all support of the SOFTWARE PRODUCT and to alter prospectively the prices, features, specifications, capabilities, functions, licensing terms, release dates, general availability, or other characteristics of the SOFTWARE PRODUCT or any associated materials.

The author notifies You that the SOFTWARE PRODUCT may cease to function properly if it is tampered with in any way, notably by removal, renaming, transferring, compression, reverse-engineering, or editing of any file that it relies upon; this expressly includes the subdirectory structure where auxiliary files are stored. You acknowledge and agree that such changes are prohibited.

## SUPPORT

This SOFTWARE PRODUCT is provided as-is, without any expressed or implied user support, with the exception of the provided documentation.

### NO LIABILITY FOR DAMAGES

In no event shall the author of this SOFTWARE PRODUCT be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary or other loss) arising out of the use of or inability to use this SOFTWARE PRODUCT, even if the Author of this Software has been advised of the possibility of such damages.
The author shall be relieved of any and all obligations for any portions of the SOFTWARE PRODUCT that are revised, changed, modified, or maintained by anyone other than the author.

### HIGH RISK ACTIVITIES

The SOFTWARE PRODUCT is not fault-tolerant and is not designed, manufactured, or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical, environmental, or other damage ("High-Risk Activities"). The author and his suppliers specifically disclaim any express or implied warranty of fitness for High-Risk Activities.

**THIRD-PARTY SOFTWARE AND SITES**
The SOFTWARE PRODUCT contains third-party software. By accepting this EULA, You are also accepting the additional terms and conditions with respect to such software, if any.

The SOFTWARE PRODUCT or its documentation may direct you to third-party Web sites. The author does not control, endorse, or guarantee content, including software, data or other information found on such third-party sites. You agree that the author is not responsible for any content, associated links, resources, or services associated with a third-party site. You further agree that the author shall not be liable for any loss or damage of any sort associated with your use of third-party content. Access to these sites, if present, is provided for your convenience only.

**SEVERABILITY**
If any provision hereof shall be held illegal, invalid, or unenforceable, in whole or in part, such provision shall be modified to the minimum extent necessary to make it legal, valid, and enforceable, and the legality, validity, and enforceability of all other provisions of this Agreement shall not be affected.

**GOVERNING LAW**
This Agreement is to be governed by and construed in accordance with the laws and jurisdiction of the defending party. Service of process upon either party shall be valid if served by registered or certified mail, return receipt requested and to the most current address provided by such party. The United Nations Convention on Contracts for the International Sale of Goods shall not apply to this Agreement.

**TERMINATION**
Your license to use the SOFTWARE PRODUCT continues until terminated. This license will terminate automatically if you fail to comply with any term hereof. No notice shall be required from the author to effect such termination. You may also terminate this Agreement at any time by notifying the author in writing of termination. On termination, you must destroy all copies of the SOFTWARE PRODUCT. Your obligation to pay any accrued charges and fees shall survive any termination of this Agreement.

**ASSIGNMENT**
Neither this Agreement nor any rights granted hereunder may be sold, leased, assigned, or otherwise transferred, in whole or in part, by you, whether voluntary or by operation of law. Any such attempted assignment shall be void and of no effect without the prior written consent of the author.

**ENTIRE AGREEMENT**
This Agreement contains the entire agreement between the author and You related to the software and supersedes all prior agreements and understandings, whether oral or written. It may be amended only by a writing executed by both parties.

**Proceeding with this installation constitutes your full acceptance of all aforementioned clauses and conditions. If you do not accept these conditions, then do not install this SOFTWARE PRODUCT iimmediately discontinue its use, and uninstall it if it was previously installed.**

**[END OF LICENSE]**

## Installation

# Installation (Windows)

- Ensure you have a working **AutoIt** environment installed, preferably the latest stable release. **Eigen4AutoIt** (E4A) was developed for **AutoIt** version **3.3.12** or later; running it on earlier versions will not work. Do not bother to report such issues, as they will not be fixed; instead, upgrade your **AutoIt** environment. You can download the latest **AutoIt** version here.

- Download the latest version of ***Eigen4Autoit.7z*** from **AutoIt**'s "Miscellaneous Downloads" section (see also the dedicated thread in the **AutoIt** *Example Scripts* forum). Extract it in a sensible location (with **7-zip**, download at http://www.7-zip.org/). Extraction will create a subdirectory called ***Eigen4AutoIt***, containing ***Eigen4AutoIt.au3*** and several other files, plus subdirectories with tests, tutorials, and source code (see below).

- You do **NOT** need to install **Eigen** itself; its relevant functionality is incorporated in the dll files you just installed.

- Open ***Eigen4AutoIt.au3*** in your favourite editor and navigate to the #region Globals; this region starts somewhere in the first thousand lines. Find the definition of global variable: **$EIGEN_DLLPATH** and replace any existing content by typing or copy/pasting (within quotation marks) the full, absolute path of the directory where the dlls are stored. Do not use relative paths, do not add any filename, and do not add a trailing backslash as last character of the path string.

- Open any E4A script (or any other AutoIt script) in **Scite** (part of the **AutoIt** package); press F5 to run, F7 to compile. More info on Scite4AutoIt3 is here.

# Installation (Linux)

- Install Wine (free, open source) for your Linux OS (currently supported are: Ubuntu, Debian, CentOS, RedHat, Fedora, SUSE, Slackware, and FreeBSD). **Wine** supports **AutoIt3** and **Scite** (the script editor that also drives the **AutoIt** interpreter and compiler).

- Install the latest stable release of **AutoIt** (download the installer, right-click it, and select "Run with Wine" or similar command). During installation, you may have to edit the installation path manually to: "C:\Program Files\AutoIt3"; the C-drive is mapped by Wine), When prompted whether to associate "Edit script" or "Run script" with double-clicking a file with .au3 extension, choose "Edit script."

- Follow the steps for the Windows installation above, but use free command-line utility p7zip to unzip the ***Eigen4AutoIt*** (E4A) package. This utility is simply the POSIX/Unix version of 7za.exe, made by the original developers of **7zip**. The E4A package can be installed anywhere; it does not have to be placed in **Wine**'s virtual C-drive. Just make sure that global variable **$EIGEN_DLLPATH** in ***Eigen4AutoIt.au3*** is edited to reflect your chosen path.

- Caveat 1: External E4A graphics-intensive utilities that rely on **Irrlicht** (such as the Function Selector tool) may not function or produce unexpected results. This restriction does not affect the core computing capabilities of E4A itself.

- Caveat 2: If you wish to use internal **AutoIt** functions and macros, those involving Windows Controls and other native Windows architecture likely won't work, or may produce unexpected results. **Wine** is not a virtual machine/emulator, so you'll have to figure out yourself what works in your environment and what does not.

# Eigen4AutoIt Directory Structure

Extraction of the compressed archive creates a directory structure (wherever you desire) summarized in the following table. No other files are added or modified.

| Files and Directories | Description |
|---|---|
| **(Top-level files)** | |
| Eigen4AutoIt.au3 | The main Eigen4AutoIt wrapper library |
| EigenDense.dll | Eigen4AutoIt's dynamic link library used in Computing mode |
| EigenDense_Debug.dll | Eigen4AutoIt's dynamic link library used in Debug mode (default) |
| Eigen4AutoIt.chm | This help file |
| Eigen4AutoIt-QuickStart_mantual.pdf | The Quickstart manual (plus a Russian translation thereof, by <ValeryVal>) |
| MatrixDisplay.au3 | external display function adapted from _ArrayDisplay() in Array.au3 (see Credits) |
| MatrixFileConverter.au3 | Import / Export utility to transfer data between matrix files and ASCII delimited, Excel, and Xbase files |
| Xbase.au3 | auxiliary for MatrixFileConverter.au3 |
| E4Aarrays.au3 | auxiliary for FunctionSelector.au3 |
| E4Aconstants.au3 | auxiliary for Eigen4AutoIt.au3 |
| E4Afuncs.au3 | auxiliary for Eigen4AutoIt.au3 and FunctionSelector.au3 |
| E4Acomplex.au3 | complex parts wrappers included in Eigen4AutoIt.au3 |
| FunctionSelector.au3/exe | external E4A function lookup utility |
| MatrixViewer.au3/exe | external E4A MatrixList and 3D real-time viewing utility |
| MV.au3/exe | Irrlicht viewport for MatrixViewer |
| bitmapfont.bmp | auxiliary for MatrixViewer.au3 |
| templates.zip | auxiliary for FunctionSelector.au3 |
| Irrlicht.dll | auxiliary for E4A/Irrlicht graphics utilities |
| IrrlichtWrapper.dll | auxiliary for E4A/Irrlicht graphics utilities |
| **.\EigenTest** | |
| EigenTest_1_Set.au3 | Test / demo of Set functions |
| EigenTest_2_Copy.au3 | Test / demo of Copy functions |
| EigenTest_3_IsCwiseEqual.au3 | Test / demo of CwiseEqual functions |

| | |
|---|---|
| EigenTest_4_MatrixSpecs.au3 | Test / demo of MatrixSpecs functions |
| EigenTest_5a_Cwise.au3 | Test / demo of Cwise functions |
| EigenTest_5b_Cwise_subset1.au3 | Test / demo of Cwise functions |
| EigenTest_5c_Cwise_subset2.au3 | Test / demo of Cwise functions |
| EigenTest_6_Transformation.au3 | Test / demo of Transformation functions |
| EigenTest_7_Multiplication.au3 | Test / demo of Multiplication functions |
| EigenTest_8_Statistics.au3 | Test / demo of Statistics functions |
| EigenTest_9a_Decomp_LowerUpper.au3 | Test / demo of LowerUpper decomposition |
| EigenTest_9b_Decomp_HouseholderQR.au3 | Test / demo of HouseholderQR decomposition |
| EigenTest_9c_Choleski.au3 | Test / demo of Choleski decomposition |
| EigenTest_9d_JacobiSVD.au3 | Test / demo of JacobiSVD decomposition |
| EigenTest_9e_Decomp_Helper.au3 | Test / demo of helper decompositions |
| EigenTest_10_File_IO.au3 | Test / demo of File I/O functions |
| EigenTest_11_TypeConversion.au3 | Test / demo of Type Conversion functions |
| EigenTest_12_ConditionalCwise.au3 | Test / demo of Conditional Cwise functions |
| EigenTest_13_Swap.au3 | Test / demo of Swap functions |
| EigenTest_14_Complex.au3 | Test / demo of complex-related functions |
| EigenTest_15_EigenSolvers.au3 | Test / demo of real and complex EigenSolvers |
| **.\EigenTutorial** | |
| EigenTutorial_1_Basics.au3 | Mode switching, elementary matrix management, and precision control |
| EigenTutorial_2_Slicing_and_Dicing.au3 | matrix cut-and-paste operations |
| EigenTutorial_3_Regression.au3 | How to fit a linear model (line, curve) to data points, using LeastSquares and linear solvers |
| EigenTutorial_4_PCA.au3 | Principal Components Analysis, from do-it-yourself building blocks to in-built functions |
| EigenTutorial_5_PCF.au3 | Principal Components Fitting, from do-it-yourself building blocks to in-built function |
| PCFdata.bin | example data set for the PCF tutorial script |
| PostOffice_E4A_Solo.au3 | Pool environment IPC hub; either compile or run in a separate instance of Scite |
| E4Amaster_Solo.au3 | Multi-Processing E4A example script |
| E4Aslave.au3 | E4A slave processes; either compile or run in separate instance of Scite |
| **.\source** | |
| EigenDense.cpp | real float part of the dll; other matrix types duplicate this with different mappings, parsed parameter types, and subset member functions |

| | | |
|---|---|---|
| | EigenDense.h | headers for the real float part of the dll |



## Credits

# Credits

## Eigen4AutoIt and EigenDense dll Source Code and Documentation

• A.R.T. Jonkers <RTFC>

## Supplementary Source Code and Documentation

• the **Eigen** developers and help file authors (past and present)
• the **AutoIt** developers and help file authors (past and present)
• <ValeryVal> (for Russian translation of this Help file and the QuickStart manual, plus various helpful suggestions for improvements)
• <randallc>, <Ultima>, Gary Frost <gafrost>, <Zedna>, <Melba23>, <AZJIO>, <UEZ>, and J.-Paul Mesnage <jpm> for _ArrayDisplay() template for ***MatrixDisplay.au3***
• <Beege> for RESHv3.1, included in the ***Function Selector*** tool
• J.Rowe <JRowe>, Andreas Templin <linus>, <ProgAndy>, and <Smashly>, for au3irrlicht2, included in the ***Function Selector*** tool
• Frank Dodd, for Irrlichtwrapper.dll, used in the ***Function Selector*** tool and the ***MatrixViewer*** tool
• <Yashied>, for ColorPicker.au3, used in the ***MatrixViewer*** tool
• <Aktonius>, for _GrabClientArea, used in the ***MatrixViewer*** tool

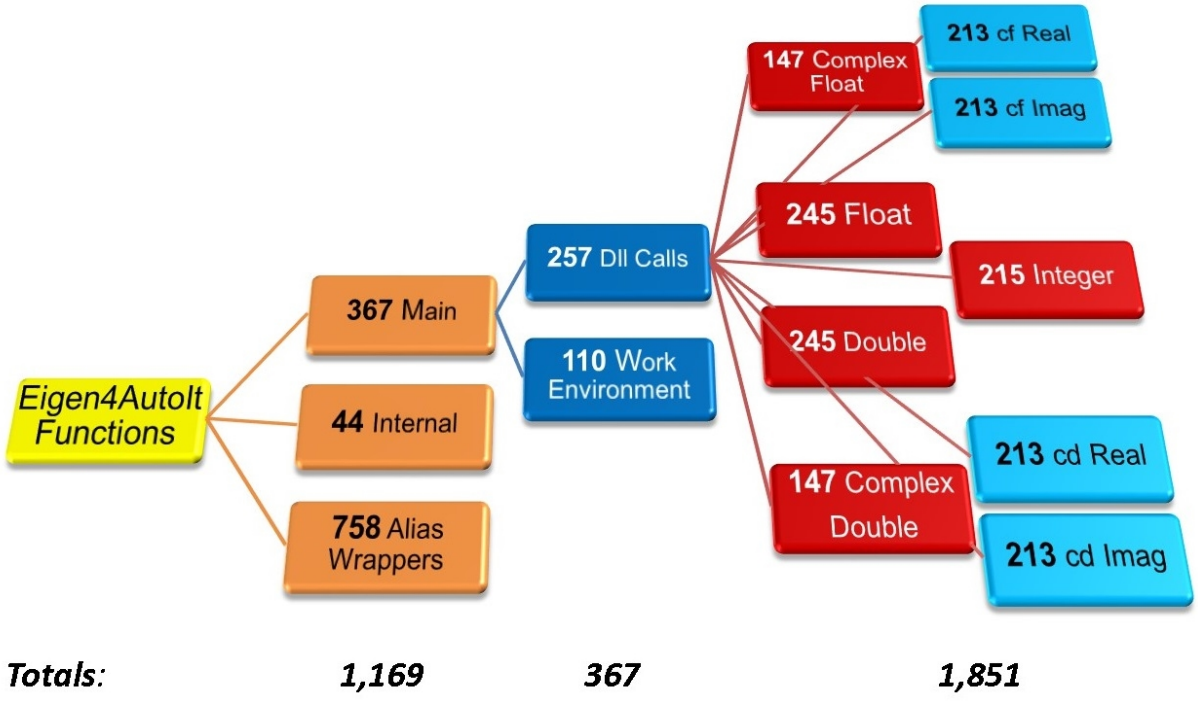
## Miscellaneous Images and Data

• **AutoIt** logos, from the AutoIt website
• **Eigen** logo, from the Eigen website
• L.I. Smith: PCA Tutorial outline and sample dataset
• L.J. Cobden, sample dataset in PCF Tutorial
• Stanislav Sýkora, mathematical and scientific constants (updated to: 2012)
• National Institute of Standards and Technology (NIST, U.S.A.), scientific constants (updated to: 2010)
• plots and graphs generated by the author in Minitab and Excel
• image of initial "O" in Tutorial PCF: Basel BS, Anatomisches_Museum, De Humani Corporis, Darstellung der antiken Kochmethode. (out of copyright, obtained via WikiMedia Commons)
• all other images: © A.R.T. Jonkers, 2014-15.

## Special Thanks

• Everyone who continues to use and support **Eigen4AutoIt**
• Everyone who continues to use and support **AutoIt**
• Everyone who continues to use and support **Eigen**

## History / ChangeLog




# History

Summarised evolution of **Eigen4AutoIt**:

**v2.5 (27 December 2015) (Release)**
•Added: function **_Eigen_PartialLeastSquares()**, which performs a Least-Squares fit in eigenspace (i.e., it is insensitive to parameter covariance and null space); this function replaces/wraps _Eigen_PCF(), which itself has undergone some internal changes to accommodate this option.
•Added global $EIGEN_LASTERROR retains the most recent **E4A** error for user-defined error handling.
•Added: global $EIGEN_ABORTEXITS allows script to return from a fatal error if set to **False**, enabling user-defined error handling within larger applications.
•Added: flag $FLAGS_PLS_Normalise (default: True); if False, PCF skips the final normalisation of results matrix V and multiplication by R-squared, yielding instead the coefficients for each original variable (= Linear single-pass Partial Least Squares). In this case, the fitted intercepts are returned separately in Rowvector **Z**.
•Added: flag $FLAGS_PLS_AddIntercept (default: True); if False, LSQ fitting omits adding an extra unity parameter column with which to fit intercept(s)
•Added **_Eigen_RsqAdjusted()**, providing a separate test statistic (in subsection Advanced) that takes into account potential overfitting of parameters
•Changed: **Eigen** includes for dlls have been upgraded to stable release version **3.2.7**; this version no longer supports computation of the determinant of integer-type matrices (see MatrixSpecs)
•Fixed: using a weighting matrix in PCF or PLS now forces $lowerbound=0, as otherwise its required dimensions are unknown at call time
•Fixed: MatrixFileConverter did not always force floating-point output when $ConvertScientificNotation=False

**v2.4 (20 May 2015) (Release)**

•Added: **MatrixViewer** utility, to display real-time animated, 3D matrix bar graphs, using the Irrlicht engine.
•Added: multi-processing support
•Added: Tutorial Pooled Multi-Processing (see E4Amaster.au3, E4Aslave.au3, and PostOffice_E4A_Solo.au3 in subdir .\EigenTutorial). Note: this requires installation of the **Pool** environment (by yours truly; included)
•Added: **_Eigen_CreateOutput_FromDims()** and **_Eigen_CreateOutput_FromIDs()**, for generating output containers for a speficied function before performing the operation (mainly useful in a multi-processing environment, where one **E4A** process may create a results matrix and another **E4A** process can fill it later.
•Added: optional UDF defined in new Global $EIGEN_ML_HASCHANGED_UDF can be called upon any significant change in the $MatrixList array
•Changed: the $MatrixList array now contains an additional fifth column, containing the name of the **E4A** function that created the matrix
•Changed: **Eigen** includes for dlls have been upgraded to stable release version **3.2.4**
•Changed: dll valid path check moved from section Globals to **_Eigen_StartUp()**, to enable applications to redefine this locally on-the-fly.
•Changed: dlls are now compiled with RunTime Libraries flags /MT /LD instead of /MD, so as to compile against the static runtime. This implies that **AutoIt** executables compiled with **Eigen4AutoIt** will no longer require access to *any* external libraries (other than the EigenDense dll(s) themselves).
•Fixed: _Eigen_OuterProduct() conditional inputs swapping (colvector-rowvector, to be supplied to dll in that order) was not fit for purpose.
•Fixed: various example code snippets corrected in Help document.

**v2.3 (21 December 2014) (Release)**

•Added: multiplication functions _Eigen_Multiply_ABCDE() plus _Eigen_Multiply_ABCDE_InPlace(), and _Eigen_Multiply_A_BdivC() plus _Eigen_Multiply_AB_divC()
•Added: _Eigen_Add_ExistingMatrix() for adding externally-defined matrix structs to the $matrixList array; mainly useful in a multi-processing environment (currently under development).
•Added: internal function _Eigen_GetNewMatrixID(), to enforce a single policy for enlarging the $matrixList.
•Changed: all Cwise functions have been **duplicated** to return either a separate results matrix **R** (the new default(!)) or to act in-place, when suffix **_InPlace** is added (formerly, this latter action was the default, but without the suffix, which was inconsistent). THIS IS A SCRIPT_BREAKING CHANGE following a suggestion by ValeryVal.
•Changed: the aforementioned change in Cwise functionality also affects conditional Cwise functions: _Eigen_ConditUnaryOp(), _Eigen_ConditScalarOp(), and _Eigen_ConditBinaryOp(). THIS IS A SCRIPT_BREAKING CHANGE.
•Changed: the aforementioned change in Cwise functionality also affects the associated official function-naming conventions, as formerly separate suffices _Col_Col and such like had to be compressed to _ColCol to free a slot for the optional last suffix: _InPlace. The longer function names are still supported through alias wrapper functions. For consistency, statistics functions _Eigen_Correlation_Row_Row() and _Eigen_Correlation_Col_Col() have likewise been officially renamed; alias wrappers again maintain access to the original names.
•Changed: $matrixList now keeps a margin of at least one hundred free slots at all times.
•Fixed: various errors, omissions, and typos in this Help document.

**v2.2 (28 November 2014) (Release)**

•Added: the ***Function Selector*** tool, an interactive 3D-animated function lookup utility, based upon the Irrlicht engine
•Added: _Eigen_Show_Constants_Nist(), _Eigen_Show_Constants_Sykora1(), and _Eigen_Show_Constants_Sykora2(), listing details of about a thousand mathematical and scientific constants that are now part of the E4A environment; these constants were compiled by Stanislav Sýkora and the U.S.

National Institute of Standards and Technology
•Added: _Eigen_Show_AllFunctions(), listing all officially named functions, with brief description (includes non-dll functions)
•Added: _Eigen_Show_MatrixSpecs_Current() and _Eigen_Show_DecompSpecs_Current(), listing the current contents of these structs (useful after calling a MatrixSpecs or decomposition function)
•Added: IsZero/IsIdentity input filter on PseudoInverse dll code
•Added: alias wrappers
•Changed: matrix replication functions _Eigen_CreateMatrix_FromA() and _Eigen_CreateMatrix_FromA_Transposed() now support one-on-one duplication as default option when no replication parameters are parsed
•Changed: removed naming inconsistencies in use of function suffix _Except and infices _To and _Show
•Changed: official function name changed from _Eigen_Covariance() to _Eigen_Covariance_Colwise(); alias still supports shorter alternative
•Changed: official function name changed from _MatrixDisplay() to _Eigen_MatrixDisplay(); alias still supports shorter alternative
•Changed: all work environment functions now return status
•Changed: various internal functions renamed for consistency
•Changed: Fatal error messages now time-out after 15s to enable multi-job processing without permanent interruption
•Changed: all alias wrappers now conform to the same internal structure
•Changed: added _Eigen prefix to all internal functions to avoid potential conflicts with external UDFs with the same name
•Changed: all internal functions are now explicitly labelled as such
•Fixed: potential stack overflow upon auto-cleanup after unexpected exit, under certain conditions
•Fixed: several functions were still missing from the general function reference list

**v2.1 (11 November 2014) (Release)**

•Added: EigenSolver section with five EigenSolvers, plus EigenTest_EigenSolver.au3
•Added: optionally complex decompositions _Eigen_ComplexSchur() and _Eigen_Tridiagonalization().
•Added: the two Schur decompositions and RealQZ now support user-defined maxIterations parameter (like some Eigensolvers)
•Added: matrix replication functions _Eigen_CreateMatrix_FromA() and _Eigen_CreateMatrix_FromA_Transposed() (tiling)
•Added: function _Eigen_ReleaseMatrix_All_Except().
•Added: function _Eigen_PseudoInverse() based upon a faster algorithm than the one in JacobiSVD, to compute the Penrose-Moore pseudo-inverse
•Added: keyword Default support for all UDFs with default parameters.
•Added: parameter bounds checks on all *internal* functions.
•Added: internal functions _WipeDecompSpecs(), _Eigen_GetMatrixTypeSuffix(), and _Eigen_Release_Except()
•Added: decompositions _Eigen_LowerUpper() and _Eigen_HouseholderQR() now support user-defined Threshold parameter to determine whether pivots are non-zero (affects linear solvers).
•Added: functions _Eigen_ShowErrors(), _Eigen_HideErrors() and internal flag $HideErrorMsg to enable interruption-free processing (provided $HideWarningMsg is also set to True); NB fatal errors are *always* displayed.
•Changed: decomposition JacobiSVD and all EigenSolvers support boolean flag $eigenvaluesAsVector, allowing matrix **S** to be shaped either as a square matrix (with eigenvalues on the diagonal) or as a Colvector. THIS IS A SCRIPT-BREAKING CHANGE for JacobiSVD. Note: the same flag is not available in statistics functions PCA and PCF, which return the eigenvalues as a Rowvector.
•Changed: _Eigen_GetActiveMatrix() will produce an error message if attempting to retrieve a currently inactive matrix
•Changed: matrix type checks are no longer performed when copying between a complex matrix part and a real matrix. This also affects new functions Tridiagonalization, EigenSolver, and EigenSolver_Generalized.

•Changed: if $maxiter is user-defined, $decompspecs is automatically activated even if $enablespecs=False, because the container is used to parse the parameter to the dll.
•Changed: whenever $decompspecs is active, all its empty values are prefilled with -1.
•Changed: internal flag $showWarningMsg renamed $HideWarningMsg (all evaluations reversed as well)
•Fixed: Householder coefficicents vector was too large in Hessenberg decomposition
•Fixed: Penrose-Moore pseudo-inverse dimensionality issues in JacobiSVD
•Fixed: _Eigen_SetMatrixType() did not support complex types.
•Fixed: _Eigen_SaveMatrix() erroneously reported it failed to save any type of real matrix (bug introduced in v2.0).
•Fixed: E4Acomplex.au3's $funclist array was not fully alphabetised.
•Fixed: CreateMatrix_<prefill> functions with a specified matrix type differing from the current work environment's type used the wrong dll function variant suffix, which could, under certain conditions, produce a randomly-timed crash later.

**v2.0 (28 October 2014) (beta Release)**

•Added: Swap functions, plus EigenTest_Swap.au3
•Added: comprehensive support for integer and complex matrices (float and double), including file I/O (save/load/convert), and function variants that act only upon the real or imaginary parts of complex input(s), using new generic function suffices: **_Real** and **_Imag**
•Added: complex-only Copy functions: _Eigen_Copy_Areal_ToB(), _Eigen_Copy_Aimag_ToB(), _Eigen_Copy_Areal_ToBreal(), _Eigen_Copy_Aimag_ToBimag(), _Eigen_Copy_Areal_ToBimag(), _Eigen_Copy_Aimag_ToBreal(), _Eigen_Copy_A_ToBreal(), _Eigen_Copy_A_ToBimag(), plus _Eigen_Swap_Areal_Aimag()
•Added: _MatrixDisplay() now supports complex inputs
•Added: EigenTest_Complex.au3
•Added: external tool **MatrixFileConverter** now supports complex inputs and outputs, including conversion to/from split storage format
•Added: work environment functions: _Eigen_IsComplex(), _Eigen_Show_DllFunctions(), and internal function _Eigen_IsComplexType()
•Added: internal functions: _Eigen_ConvertMatrix_ToInterleaved(), _Eigen_ConvertMatrix_ToSplit(), to handle non-native complex storage
•Added: internal function: _Eigen_CheckDllCallSuffix(), to determine whether a particular function variant is supported in the dlls
•Added: internal flag $IgnoreTypeMismatch; when set, allows internal function _Eigen_GetPtr() to always pass its matrixtype check
•Added: internal work environment globals $EIGEN_COMPLEX and $EIGEN_COMPLEXREALIMAG
•Added: complex parts-accessing alias wrappers, in external include "E4Acomplex.au3"
•Changed: internal function _Eigen_GetPtr(), including different parameters
•Changed: Penrose-Moore pseudo-inverse algorithm replaced by template-optimised version in JacobiSVD dll code
•Changed: internal matrix pointer administration, affecting the large majority of functions (except work environment and decompositions); all functions that create or manipulate matrices now affect global table $ActiveMatrices and the associated bit flags in $EIGEN_ACTIVE_MATRICES
•Changed: _Eigen_CreateMatrix_fromAcols() and all similar functions: col/row index matrix **V** is now always type <int>, regardless of the work environment's matrixtype
•Fixed: all UDF-internal variables are locally declared

**v1.4 (6 October 2014) (Release)**

•Added: Tutorial Slicing and Dicing
•Added: advanced stats function _Eigen_PCF(), i.e., Principal Components Fitting, plus associated new

Tutorial PCF
•Added: stats functions _Eigen_Rsquared() (a.k.a. coefficient of determination), including _Eigen_Rsquared_Col(), _Eigen_Rsquared_Colwise() to compute the percentage of total variability accounted for by a model fit
•Added: generic goodness-of-fit functions _Eigen_RelativeError(), _Eigen_RelativeError_Col(), _Eigen_RelativeError_Colwise(), _Eigen_Misfit_Col(), Eigen_SSR_Col(), _Eigen_Misfit_Colwise(), _Eigen_SSR_Colwise()
•Added: matrix management functions: _Eigen_CreateMatrix_LinSpaced_Colwise(), _Eigen_CreateMatrix_LinSpaced_Rowwise()
•Added: matrix management functions: _Eigen_CreateMatrix_FromAblock(), _Eigen_CreateMatrix_FromAcols(), _Eigen_CreateMatrix_FromArows(), _Eigen_CreateMatrix_FromABcols(), _Eigen_CreateMatrix_FromABrows(), _Eigen_SplitMatrix_FromAcol(), _Eigen_SplitMatrix_FromArow(), plus their transposed equivalents
•Added: all Normalise functions now also support divisor <absolute total> as alternative to default <StDev>
•Added: environment functions _Eigen_Show_Warnings() and _Eigen_Hide_Warnings() (mostly-affects type-conversion user confirmation)
•Added: error handling allows user to terminate script immediately if pressing <Cancel> button
•Added: internal flag $showErrorChain allows display of additional error messages as error is parsed back up the calling chain (default: False, i.e., only the function that detected the error displays an error message)
•Added: many alias wrappers
•Changed: all alias wrappers are now empty shell functions (slower), whereas the function bearing the official name contains the real code (faster)
•Fixed: all alias wrappers now parse any errors back up the call chain
•Added: MatrixFileConverter (beta version **0.8**) export to Excel can add new worksheet to current workbook to store results
•Added: MatrixFileConverter support for importing free-format, space-delimited ASCII (fixed columns with variable spacing depending on variable length of content per cell); this conversion is now also a bit faster (redundant branching removed)
•Fixed: MatrixFileConverter trimmed path displays still did not work properly at startup

**v1.3 (21 September 2014) (Release)**

•Added: Example code snippets for all functions in Help file
•Added: CreateMatrix functions with pre-specified content: _Eigen_CreateMatrix_Constant(), _Eigen_CreateMatrix_Identity(), _Eigen_CreateMatrix_Ones(), _Eigen_CreateMatrix_Random(), _Eigen_CreateMatrix_Zero()
•Changed: _Eigen_SetMatrixType() can now initialise the first session too, or restore defaults for current type.
•Added: internal function _Eigen_SetMatrixTypeGlobals()
•Changed: moved two Copy functions to section "Copy"
•Changed: minor edits to Tutorials and Test scripts
•Fixed: bug in ConvertMatrixFile dll code
•Fixed: many broken links, typos, and other mistakes in the Help text
•Fixed: Eigen's assert failure messages were not displaying properly in some environments

**v1.2 (18 September 2014) (Release)**

•Added: this Help file (no example code snippets yet)
•Changed: **Eigen** includes for dlls upgraded to release version **3.2.2**
•Changed: _Eigen_CwiseBinaryOp() and all its subfunctions had their parameter order changed so the last two are always: $operator, $matB. THIS IS A SCRIPT-BREAKING CHANGE.

•Changed: the large majority of functions now all behave in the same way in returning either (non-zero) target matrix ID or True upon success and False + nonzero @error flag upon failure. This affects for example all decompositions, which formerly returned an Eigen status code (now incorporated into error message upon failure). Notable exceptions are: all work environment functions (no return value), Show_<some list> functions (no return value), and functions that return a single value directly. THIS IS A SCRIPT-BREAKING CHANGE.
•Changed: various functions formally moved between regions work environment and matrix management (no change to functionality)
•Changed: _Eigen_GetMatrixVarType renamed to _Eigen_GetMatrixTypeID() (original name still supported trhrough alias wrapper)
•Changed: major function sections reformatting (underscores, naming conventions, suffix order, alternatives; original versions still supported through alias wrappers); this affects about 20% of all functions. For example, all decompositions now start with the prefix _Eigen_Decomp_, and type conversion functions all look like this: _Eigen_ConvertMatrix_ToDouble().
•Changed: _Eigen_DebugMode_On() and _Eigen_DebugMode_Off() now have an optional parameter to keep existing matrices
•Changed: all Tutorial scripts and all test scripts to use the new, standardised nomenclature
•Changed: minor edits in Quickstart manual
•Fixed: missing bounds checks added to all LinSpaced functions.
•Added: display lists of environment globals, defined matrices: _Eigen_Show_EnvironmentVars(), _Eigen_Show_MatrixList()
•Added: display lists of specs: _Eigen_Show_MatrixSpecs(), _Eigen_Show_DecompSpecs()
•Added: internal function: _BitClear()
•Added: MatrixList control through direct indexed access: _Eigen_ReleaseFromIndex_ToIndex(), _Eigen_ReleaseFromIndex_ToIndex_Except, ReleaseFromIndex(), ReleaseFromIndex_Except()
•Added: selective matrix deactivation: _Eigen_ResetActivematrix_Single()
•Added: many, many new alias wrappers

**v1.1 (29 August 2014) (Release)**

•Added: display lists of operators: _Eigen_Show_BinaryOperators(), _Eigen_Show_ConditOperators(), _Eigen_Show_ScalarOperators(), _Eigen_Show_UnaryOperators()
•Added: alias wrappers
•Fixed: various inconsistencies between AutoIt wrapper functions and dll functions called (mainly concernning parsed parameters and returned results)
•Fixed: MatrixFileConverter GUI display of long filenames was trunctated incorrectly

**v1.0 (19 August 2014) (Release)**

•Added: external matrix file importer/exporter tool: ***MatrixFileConverter*** (GUI / cmdline with parameters)
•Added: alternative Transpose for matrix files: _Eigen_RedimExistingMatrixFile()
•Added: alias wrappers

**v0.9 (1 August 2014) (beta Release)**

•Added: conditional Cwise operators: _Eigen_ConditUnaryOp(), _Eigen_ConditScalarOp(), _Eigen_ConditBinaryOp()
•Added: conditional evaluation: _Eigen_ConditAll(), _Eigen_ConditAny(), _Eigen_ConditCount()
•Added: MatrixList matrix properties retrieval: _Eigen_GetMatrixCols(), _Eigen_GetMatrixRows(),

_Eigen_GetMatrixType(), _Eigen_GetMatrixVarType(), later renamed _Eigen_GetMatrixVarID()
•Added: alternative Transpose: _Eigen_RedimExistingMatrix()
•Added: Test script Conditional Cwise
•Added: alias wrappers

**v0.7-0.8 (21 July 2014) (beta Releases)**

•Changed: cleaned up the Eigen4Autoit.au3 code and header
•Added: alias wrappers
•Fixed: _ArrayDisplay() parameters in various calls

**v0.6 (21 July 2014) (first beta Release)**

•Created: **Eigen4AutoIt** wrapper library
•Created: EigenDense_Debug.dll, based on **Eigen** release **3.2.1**
•Created: EigenDense.dll (ditto)
•Created: external function *_MatrixDisplay()*, adapted from _ArrayDisplay() in standard include file Array.au3 (see Credits for acknkowledgements)
•Created: Tests scripts 1-11
•Created: Tutorial scripts 1-3 (Basics, Regression, PCA)



## Tutorials

# Tutorials

- Basics
- Slicing and Dicing
- Regression (fitting a function to points)
- Principal Components Analysis (PCA)
- Pooled Multi-Processing



## Basics

# Tutorial - Basics

Welcome to the AutoIt-Eigen matrix computing environment called **Eigen4AutoIt**!

This tutorial explains the basics of **Eigen4AutoIt**: operational modes, elementary matrix management, and precision control. The tutorial assumes that you have already installed AutoIt and **Eigen4AutoIt**, that you know how to create, edit, and run **AutoIt** scripts, and that you are somewhat familiar with AutoIt functions in general (parameter parsing, optional arguments, and default

arguments). But even if you start from zero, this Tutorial should be easy to follow. **E4A** was specifically designed to make matrix computing intuitive and accessible.

You can load the full script of this tutorial in Scite (or your own favourite editor) and/or run it alongside while reading. It is located in **Eigen4Autoit**'s subdirectory ***EigenTutorial***. In Windows, a full AutoIt install will allow you to right-click any AutoIt script filename, and select the option "Edit Script," which will open it in the Scite script editor, from where you can start running the interpreter by pressing **F5**. Any errors will be listed in the output window at the bottom (opened after F5 is pressed the first time). If a script completes without any problems in the interpreter, you can then compile it into a stand-alone executable by pressing **F7** in Scite. See also AutoIt's documentation for extensive details on various other ways to do this.

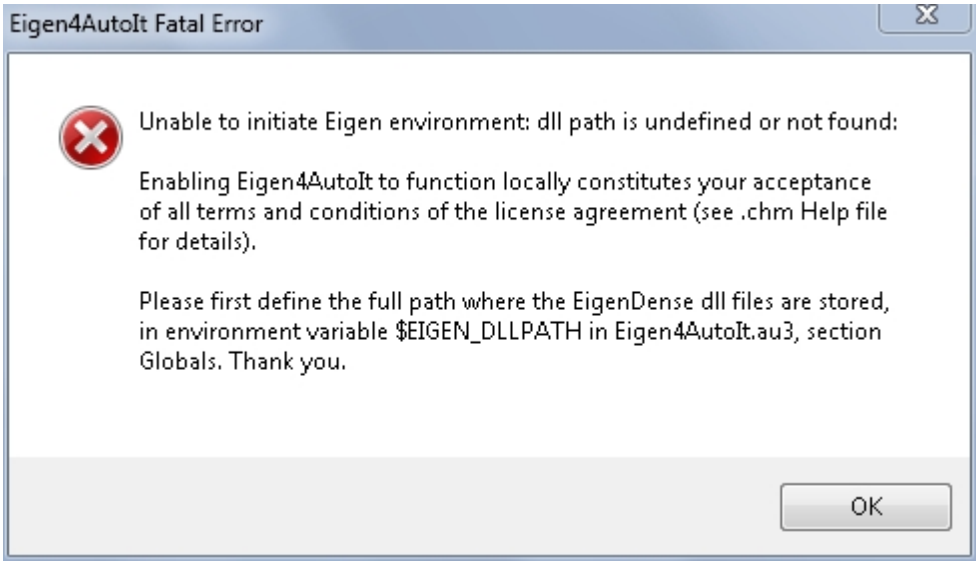
Let's roll. To enable the **Eigen4AutoIt** environment in your own scripts, you'll need to include the wrapper library ***Eigen4AutoIt.au3***, so somewhere near the top of such a script you have to add this line:

```
#include "Eigen4AutoIt.au3"
```

The hash symbol ("#") at the start means this line is not executable code, but a compiler instruction; in this case, your'e telling the AutoIt compiler to insert the entire contents of the script file ***Eigen4AutoIt.au3*** to your current script. This particular line assumes you've placed this file in the local directory where your script will run or be compiled. If it is stored somewhere else, and your set-up does not provide a permanent path to that location, you'll have to include the **full** path in the #include compiler directive. So if ***Eigen4AutoIt.au3*** lives in subdirectory *C:\AutoIt\maths\matrix*, for example, then you would write:

```
#include "C:\AutoIt\maths\matrix\Eigen4AutoIt.au3"
```

If this line were your entire script, the very first time you attempted to run it, you *might* get a **fatal** error message complaining that it cannot locate the dlls (the dynamic link libraries, which contain all of Eigen's matrix magic):




You have to set the full path of the dlls explicitly, in the associated environment variable **$EIGEN_DLLPATH** in ***Eigen4AutoIt.au3***. Please consult the installation instructions here if this happens. You only have to do this once, after installation.

The environment contains two versions of the ***EigenDense*** dll. Why two instead of just one? The **Debug** version features extensive error checking, both in the dll and inside the various Eigen modules it calls.

There are many unforeseen reasons why some computation you're developing may fail. In **Debug** mode (the default mode, which uses ***EigenDense_Debug.dll***), many of these circumstances will be caught and reported, slowing processing down. Please do not use this version for benchmarking.

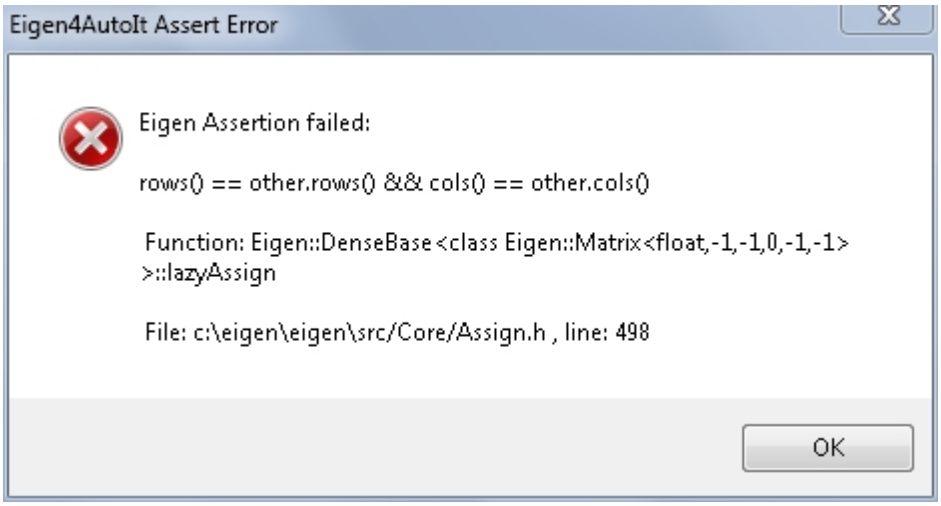
Let us start by testing this switching between **Debug** mode and **Computing** mode. To do this, we'll need to initialise an **Eigen4AutoIt** work environment, like so:

```
_Eigen_Startup()
```

By default, **_Eigen_Startup()** will run in **Debug** mode, meaning **Eigen** will trigger so-called assert messages if various (error and fail) conditions occur. Normally, the AutoIt wrappers in **Eigen4AutoIt** would catch some of these, before the call ever reaches **Eigen**, so for demonstration purposes we'll use a special test function to force an assert failure, by attempting to perform an illegal operation, namely, transposing a non-square matrix inPlace (which is very, very naughty):

```
MsgBox ( 0, "Do not be alarmed", "An Eigen Assert is supposed to be triggered when <OK> is pressed." )

_Eigen_Test_EigenAssert ()

MsgBox ( 0, "Test_EigenAssert", "After the assert-triggering call" & @CR & "(if all went well, an assert error message appeared)." )
```

Between the two MsgBox windows, this error message should have been generated from within ***EigenDense_Debug.dll***:




Assert errors may look a bit cryptic at first. If you get these in your own scripts, the important line is the second one. In this case, it tells you that the dll encountered some kind of mismatched dimensions problem, which is exactly what happens when you try to squeeze a rectangular (non-square) shape into the same shape mirrored along the diagonal (so the unequal row and column sizes are exchanged, see transposition).

Now we'll switch from **Debug** mode to **Computing** mode, and do the same test again:

```
_Eigen_DebugMode_Off ()        ; alternatively, we could have used _Eigen_ComputingMode_On(), for example

MsgBox ( 0, "Test_EigenAssert", "Now we've switched to Computing mode." & @CR & "If we try our test assert call again, nothing happens..." )

_Eigen_Test_EigenAssert ()        ;  the error still happens inside the dll call, but no message is triggered!

MsgBox ( 0, "Test_EigenAssert", "After the assert-triggering call" & @CR & "(if all went well, NO assert error message appeared this time)." )
```

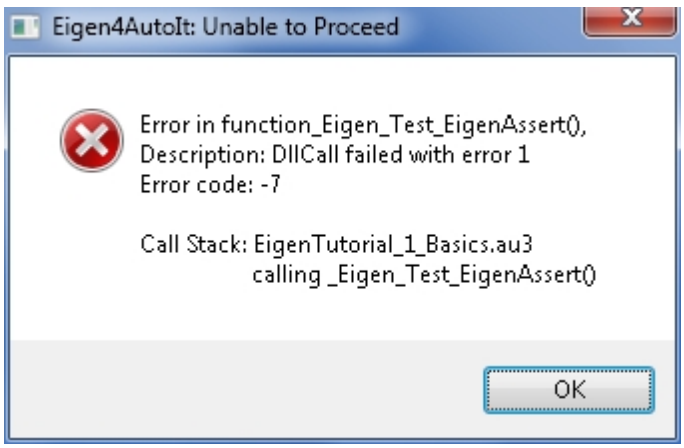
This time around, no failed assert message appears, *even though the error still occurs*! This highlights the importance of **Debug** mode, which checks all kinds of potential pitfalls and mistakes before performing any operation (far more than the wrapper functions in ***Eigen4AutoIt.au3*** can do). Whenever you develop new code, it makes sense to keep running in **Debug** mode until no such assert failures are triggered under a wide variety of test conditions. The price you pay is performance loss (which should be unimportant at the developing stage). Only once you are certain your code is rock solid should you switch to **Computing** mode, which is blazingly fast, but does no error checking once the dll is called, so *any* glitch during the computation itself will likely produce complete garbage as final output. You have been warned.

Another aspect worh noting is the comment in the first line of the last code snippet. It states that instead of using **_Eigen_DebugMode_Off()**, we could have called a different function, **_Eigen_ComputingMode_On()**, with the exact same outcome. Functions such as **_Eigen_ComputingMode_On()** are so-called **alias wrappers**; they wrap the true function (in this case, **_Eigen_DebugMode_Off()**) inside a "renaming cloak," that simply calls the real function (with any arguments parsed along) and returns whatever the real function returns. Few things are more annoying than having to look up the exact way a particular function should be expressed, so wherever possible, one or more obvious alternatives have been included as transparent aliases, to make your life easier. So for the earlier example, other alternatives include **_Eigen_SetComputingMode()**, **_Eigen_SetComputingMode_On()**, **_Eigen_ResetDebugMode()**, **_Eigen_SetDebugMode_Off()**, **_Eigen_SetDebugOff()**, **_Eigen_SetDebugModeOff()**, and **_Eigen_Debug_Off()**.

A final issue you might encounter is forgetting to initialise the work environment altogether. What if the entire script consisted of the following single line, without first (directly or indirectly) calling **_Eigen_Startup()**:

```
_Eigen_TestAssert ()
```

Then regardless of which **Eigen4AutoIt** function you attempted to call, you would encounter this error message:



The first error in this Tutorial was a Fatal error, the second one was an assert failure inside the dll; but this third type of error message is the one you'll see most. It states:

- which **Eigen4AutoIt** function detected the error;
- a brief description of the error; in this specific case, it also provides the (positive) error code that the DllCall itself generated; if you check the AutoIt documentation for DllCall, you'll find that @error = 1: "unable to use the DLL file";
- the (always negative) **Eigen4AutoIt** error code: **-7**; you can look up its meaning in the appendix here;
- the Call Stack; this can be quite useful for tracing a problem to its source in a long chain of deeply-nested **Eigen4AutoIt** calls. In this example, the stack is only two levels deep: the script at the top (1), calling the test function (2).

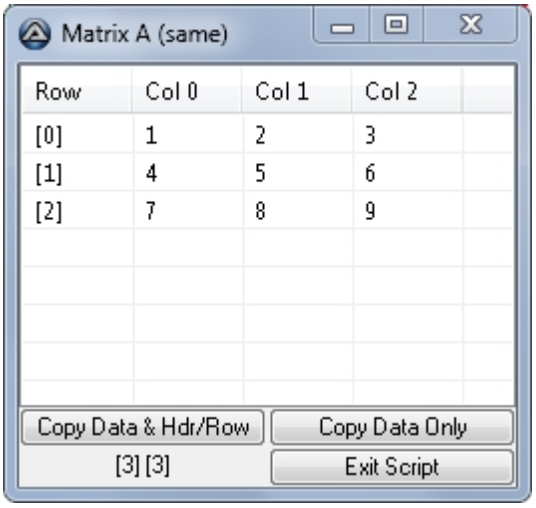
Well, this is going swimmingly! We've already generated three major errors in our first Tutorial. Well done!

Let's try to do some real work now, and get to grips with matrices. We will initialise an **Eigen4AutoIt** environment again in debugging mode, fill an ordinary AutoIt array with some data, and then create a matrix variable from that array:

```
_Eigen_DebugMode_On ()        ; this function calls _Eigen_StartUp() with Debug settings

Local $arrayA[3][3] = [ [1,2,3], [4,5,6], [7,8,9] ]    ; create a two-dimensional, 3 x 3 array with some data

_ArrayDisplay ( $arrayA, "Array A" )       ; Eigen4AutoIt.au3 contains the line: #include <Array.au3>

$matA =_Eigen_CreateMatrix_FromArray ( $arrayA, True )          ; (default) True = copy data too

_MatrixDisplay ( $matA, "Matrix A (same)" )       ; Eigen4AutoIt.au3 contains the line: #include
"MatrixDisplay.au3"
```

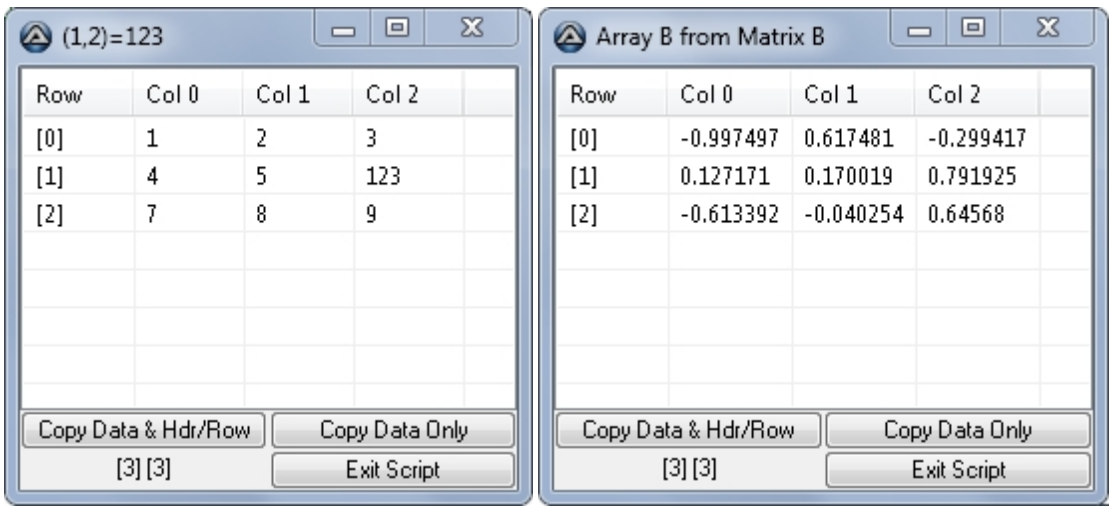
A matrix is a 2D data grid, like an array in some respects, but 128-bit aligned, and contiguous in memory. Matrix memory locations (the equivalent of array elements) are often called "cells" or "coefficients;" they are all of the same type (float, double, int, ...). In this environment, float (4 bytes) is the default, with default precision: **1.0e-6**. Double precision is discussed further down in this Tutorial.

The external function **_MatrixDisplay()** is virtually identical in usage to _ArrayDisplay(); please consult the latter function's documentation for the various options (such as displaying some specific part of a large matrix). A matrix is by definition two-dimensional (2D) in this environment. In the example below we have a 3 x 3 (rows x columns) matrix filled with the values **1** through **9**. Note that the cell at coordinates 1,2 (order: [row, column], base-0) contains the value **6** at the moment. These values may look like integers, but are internally stored as floats. It's worth keeping in mind that regardless of the actual number of bits used to store values, there will always be some values that cannot be *exactly* represented. The gaps between representable values increases with (positive or negative) distance from zero. This is a limitation of all binary computing, just as the value 1/3 has to be truncated at some decimal when written down (0.333333333333... ad infinitum). Nevertheless, double precision suffices for most real-world applications.




Let's do some basic cellwise I/O (input/output), then copy the matrix, fill the duplicate with random values and copy those data to another array:

```
MsgBox ( 0, "Reading (1,2)", _Eigen_ReadMatrixValue ( $matA, 1, 2 ) )      ; coordinate order is: row, column

MsgBox ( 0, "Writing 123", _Eigen_WriteMatrixValue ( $matA, 1, 2, 123 ) ) ; nested call

_MatrixDisplay ( $matA, "(1,2)=123" )

MsgBox ( 0, "Reading (1,2)", _Eigen_ReadMatrixValue ( $matA, 1, 2 ) )      ; reading the new value

$matB = _Eigen_CloneMatrix ( $matA, False )      ; False = do not copy data across (default is True)

_Eigen_SetRandom ( $matB )     ; fill matrix B with some random values in  the range: (+/-) 0-1

_MatrixDisplay ( $matB, "random Matrix B" )

Local $arrayB  ; $arrayB has to be a declared variable, but not necessarily an array, and not necessarily of the correct dimensions yet

_Eigen_CreateArray_FromMatrix ( $matB, $arrayB ) ; $arrayB is here re-created as a correctly-sized container to hold the data in matrix A

_ArrayDisplay ( $arrayB, "Array B from Matrix B" )
```



In the first two lines, note how the **Eigen4AutoIt** function is embedded into another function (MsgBox) as argument. Most functions that change a matrix or produce a new "results" matrix (often called $matR) will, upon success, return that matrix's unique ID (explained here), so it can immediately be used as parameter to parse to another **Eigen4AutoIt** function. Complicated operations requiring multiple nested steps can thereby be compressed into a single line of code, without the need to store the matrix ID of each intermediate result in a separate variable.

We can also create matrices from scratch and use them as output containers. In the next code snippet, we will create a 3 x 3 results matrix **R** , and fill it with zeroes, using a different Set function called "SetZero." Then we'll matrix-multiply matrices **A** and **B**, and use our pre-existing, correctly-sized results matrix **R** to capture the result. Alternatively, if we don't supply an existing container, a new output matrix (**C**) is created when we multiply, which allocates additional memory. If we then decide to copy those data to a new **AutoIt** *array* (a slow process you'll want to avoid as much as possible), we basically duplicate the data, and use up even more memory resources. Finally, we release our matrices from memory again.

```
$matR = _Eigen_CreateMatrix ( 3, 3 )     ; creating matrix R, with dimensions: 3 rows by 3 cols
_Eigen_SetZero ( $matR )          ; fill the entire matrix with zeroes; alternatively, use
_Eigen_CreateMatrix_Zero() directly
_MatrixDisplay ( $matR, "prezeroed results R" )

_Eigen_Multiply_AB ( $matA, $matB, $matR )       ; the third, optional parameter identifies the output
container
_MatrixDisplay ( $matR, "Multiply_AB result in R" )

$matC = _Eigen_Multiply_AB ( $matA, $matB )     ; third parameter omitted = create a NEW output
container
_MatrixDisplay ( $matC, "Multiply_AB => matID nr: " & $matC )

Local $arrayC [3][3]      ; create a new, correctly-sized array C to store the contents of matrix C
_Eigen_CopyMatrixData_ToArray ( $matC, $arrayC )        ; this is a SLOW process...
_ArrayDisplay ( $arrayC, "Array C")        ; same data, different type of container

_Eigen_ReleaseMatrix ( $matA )            ; release the memory allocated for these matrices
_Eigen_ReleaseMatrix ( $matB )
_Eigen_ReleaseMatrix ( $matC )
_Eigen_ReleaseMatrix ( $matR )
```

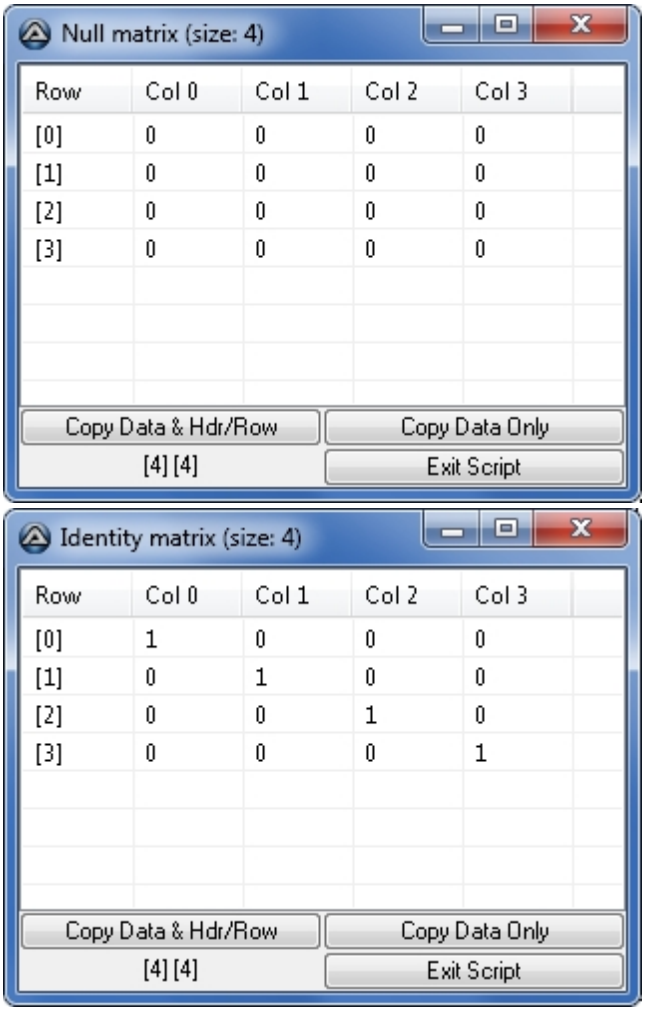
Regarding releasing memory, instead of repeating **_Eigen_ReleaseMatrix()** for each matrix, we can also do this in bulk, by placing a so-called **ListMarker** (with **_Eigen_SetListMarker()**) before we start creating new matrices, and afterwards cleaning up all matrices created after we placed our marker,with **_Eigen_ReleaseFromMarker()**. We can also exclude from this process any specific matrices we wish to keep, by calling **_Eigen_ReleaseFromMarker_Except()** instead. See Tutorial "Regression" for an example.

In summary:

- we can create empty matrices directly, with **_Eigen_CreateMatrix()**
- we can create matrices from arrays directly, with **_Eigen_CreateMatrix_From Array()**
- we can create arrays from matrices, provided the array variable has previously been declared (but not necessarily yet as an array, or as an array of the correct size); **_Eigen_CreateArray_FromMatrix()** creates an array of the right dimensions, whereas **_Eigen_CopyMatrixData_ToArray()** expects an existing array of the correct dimensions, and just copies the values
- most **Eigen** functions produce a single matrix result, by either:

    - storing it in an identified, existing output container you provide, or
    - allocating a new matrix in memory, and returning its unique matrix ID number;
- some special functions return a computed value directly, or a status code, or a boolean (True = success, False = failure). (A different class of functions that return multiple matrices simultaneously is introduced in Tutorial Slicing and Dicing.)

**Scalars** can be interpreted as 2D matrices with both dimensions being one. You know them well from ordinary, boring arithmetic. If you need larger matrices that behave arithmetically as scalars, create a square **Null matrix** of the desired size (a matrix filled entirely with zeroes) and then fill every diagonal cell with a fixed value (with **_Eigen_SetConstant_Diag()**). For example, an **Identity matrix** of any size (commonly denoted: "**I**") has the value one (1) in all diagonal cells, and is zero everywhere else; it is the matrix equivalent of a scalar value of **1**. The identity matrix is crucial in linear algebra, notably when dealing with matrix inversion. We can create one from scratch with **_Eigen_CreateMatrix_Identity()**.

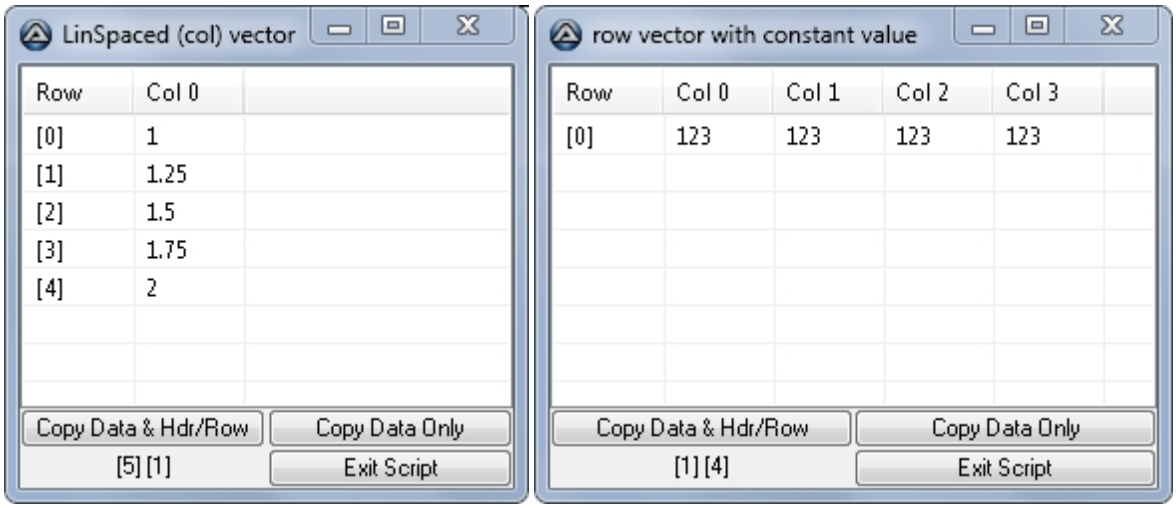



**Vectors** behave just like other matrices; the only difference is that one dimension has a size of unity (one), whereas the other dimension is larger. We distinguish two types of vector shape:

| Vector | **Rows** | **Columns** | **Shape** |
|---|---|---|---|
| **Col** vector | **>1** | **1** | vertical |
| **Row** vector | **1** | **>1** | horizontal |

Let's create some:

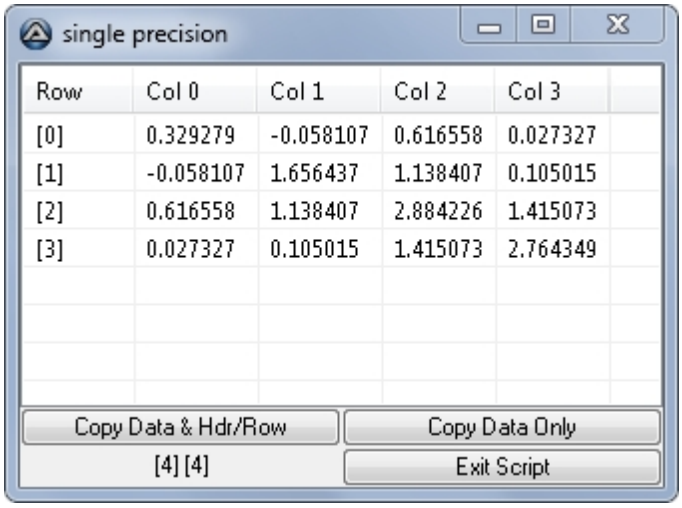
```
$colvector = _Eigen_CreateMatrix ( 5, 1 )          ; column dimension = 1

_Eigen_SetLinSpaced_Col ( $colvector, 0 ,1, 2 )    ; fill column zero with equal-spaced data in the range 1 to 2

_MatrixDisplay ( $colvector, "LinSpaced (col) vector" )


$rowvector = _Eigen_CreateMatrix ( 1, 4 )          ; row dimension = 1

_Eigen_SetConstant ( $rowvector, 123 )  ;  fill with constant value 123

_MatrixDisplay ( $rowvector, "row vector with constant value" )
```




Finally, a quick demo of precision control. Remember that by default, the **Eigen4AutoIt** environment uses floating point values ("reals") that require four bytes of storage each. This is called **single precision**. Let's define a test matrix with some random reals. Cell values in your own environment may differ from the screenshots reproduced below.

```
$matA = _Eigen_CreateMatrix ( 4, 4 )     ; create square matrix of size 4

_Eigen_SetRandom ( $matA )     ; fill with random values; alternatively, use _Eigen_CreateMatrix_Random()

_Eigen_Multiply_AAt_InPlace ( $matA )     ; do some dumb arithmetic

_MatrixDisplay ($matA, "single precision")          ; show it
```

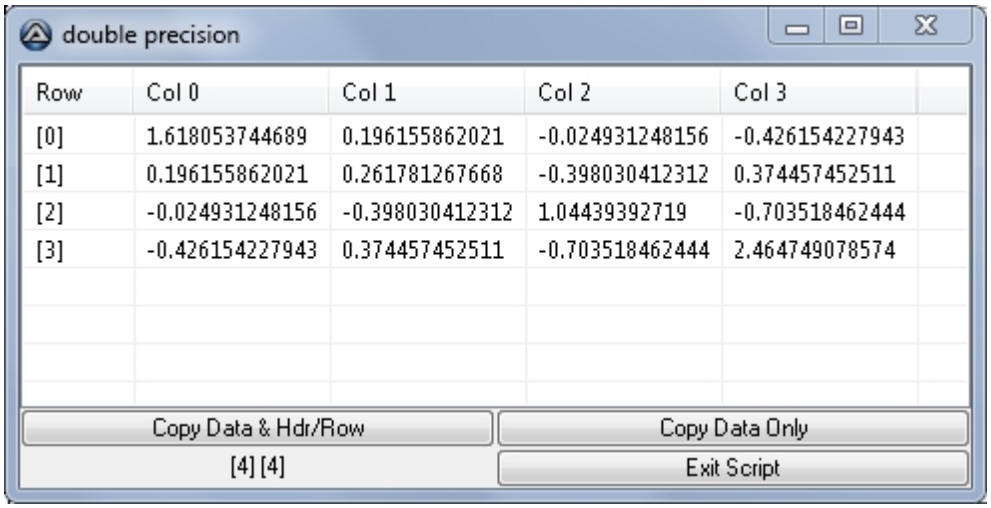
Thrilling stuff. Incidentally, this time we performed the multiplication "**InPlace**", meaning we overwrote the input matrix's contents with the output values, having become frugal with our precious memory resources.




Now we'll switch to **double precision** and repeat the exercise, and close down. As you are **not supposed to mix different matrix types**, this normally entails a clean break, meaning all pre-existing matrices are released from memory. If you do not want this to happen, you can call _Eigen_Startup ( "double", **False** ), i.e., with the optional second parameter set to **False**, but then you'll have to convert each float-type matrix to double-type before you can use it in **Eigen** calls again.

```
_Eigen_Startup ( "double" )

$matA = _Eigen_CreateMatrix ( 4, 4 )     ; create square matrix of size 4

_Eigen_SetRandom ( $matA )     ; fill with random values

_Eigen_Multiply_AAt_InPlace ( $matA )     ; I'm having a déjà-lû...

_MatrixDisplay ($matA, "single precision")          ; show it

_Eigen_CleanUp ()         ; close the dll, release all matrices from memory

MsgBox ( 0, "Eigen4AutoIt", "This concludes this Tutorial." )
```

The important difference with the previous output is not the values themselves (they are random, remember?), but their number of significant digits: a whopping *twelve* decimal places here, against a mere *six* up above. In other words, the accuracy of double precision = **1.0e-12**, at the cost of doubling the storage requirements, both in memory and on file. Whether that is a price worth paying is up to you.



## Slicing and Dicing

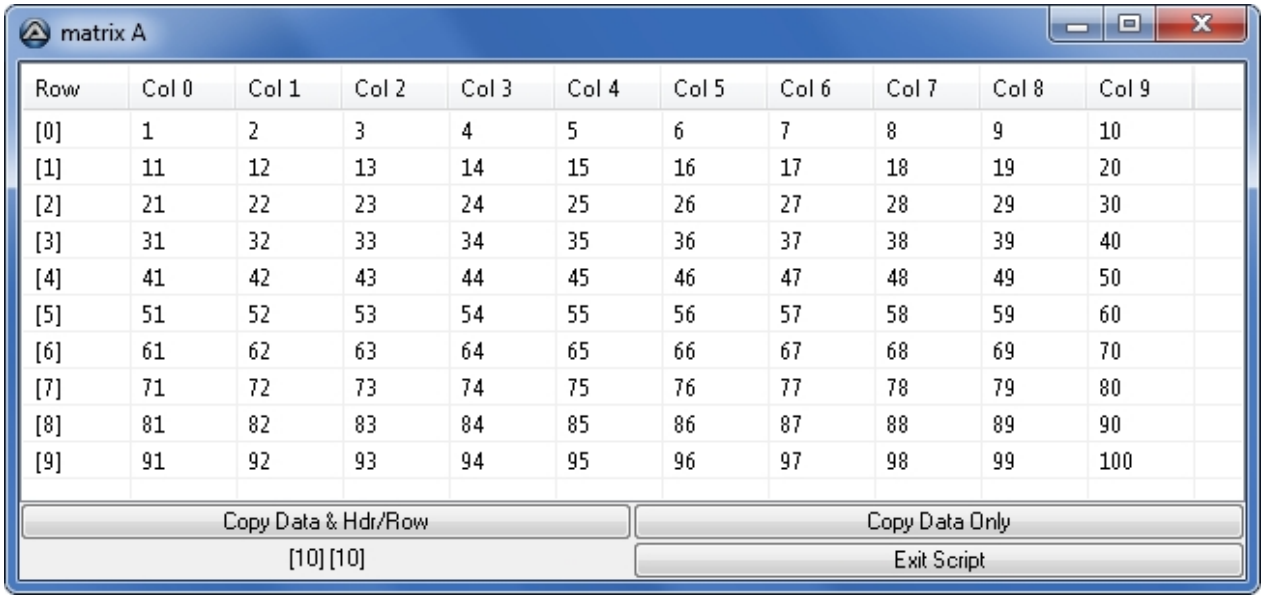
# Tutorial - Slicing and Dicing

This brief beginner's Tutorial is about repackaging matrix cells, from one existing source matrix into one or more **new** destination matrices. Unlike Copy functions, which transfer data between two **existing** matrix containers, the following cut-and-paste functions always create at least one new matrix, with dimensions that differ from those of the source matrix. You can find detailed descriptions in the Function Reference, section Matrix Management, mostly in sub-section Matrix Creation.

Slicing and dicing can be great fun; just ask your butcher, or your friendly local serial killer.

```
#include "Eigen4AutoIt.au3" ; sine qua non

_Eigen_Startup ()         ; on the road again...

; create a matrix with consecutive integers in horizontal order
$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 10, 10, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )
```

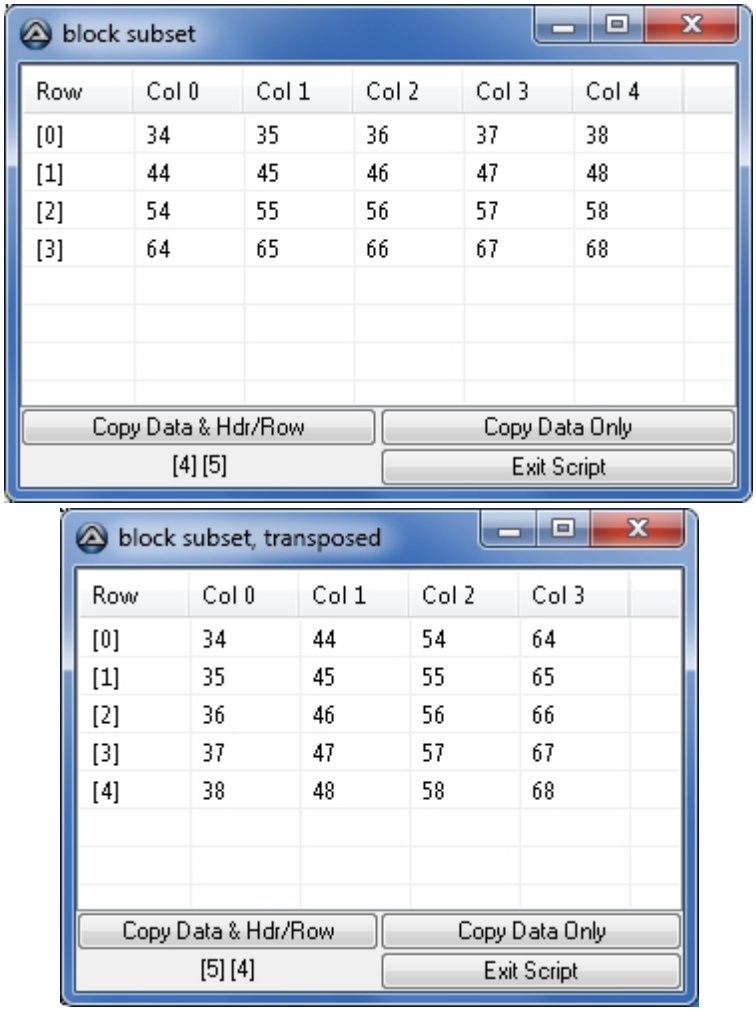
The first two lines of this script should become second nature whenever you use **Eigen4AutoIt** calls, hooking up the AutoIt **wrapper library** and the **dynamic link library** (hereafter: "dll"). The matrix creation function that follows generates a linearly-spaced sequence of numbers to fill the matrix cells with. This can be done vertically ("ColMajor") or horizontally ("RowMajor"); here we use the latter. The values themselves appear as integers (although internally stored as single-precision floats) because the first and last values of the sequence (1 and 100) define a range that yields integer steps when divided by the number of elements (rows times columns = 10 times 10 = 100). If any single one of those four parsed parameters (10, 10 , 1, 100) had been different, both the step size and the cell contents would have been non-integers ("reals"). We use integers here to make it easy to recognise which parts of the source matrix will be the basis for the new matrices we'll be creating.




Let the Chopping Commence!

```
$startRow    = 3    ; define a non-square block somewhere near the centre
$startCol    = 3
$blockRows   = 4
$blockCols   = 5

; create a new matrix from the block data
$matR = _Eigen_CreateMatrix_FromAblock ( $matA, $startRow, $startCol, $blockRows, $blockCols )
_MatrixDisplay ( $matR, "block subset" )

; repeat, but flip the block this time (mirror along its diagonal)
$matR = _Eigen_CreateMatrix_FromAblock_Transposed ( $matA, $startRow, $startCol, $blockRows,
$blockCols )
_MatrixDisplay ( $matR, "block subset, transposed" )
```

Here we define a matrix **block**, which can be any rectangular shape that fits inside the source matrix. Its boundaries may coincide with, but not extend across, any of the source matrix's boundaries. Blocks are defined with four parameters: the **top** and **left** coordinates (row and column, in that order, **base-0**), followed by the block's **height** (in rows) and **width** (in columns). This particular block has four rows and five columns (so twenty elements altogether), and its top left corner is at cell (3,3). When calling _Eigen_CreateMatrix_FromAblock(), a new matrix is created and added to the global $MatrixList array, and its new index (the matrix ID) is returned. This latest addition to the family has the same dimensions and the same cell contents as the block we just defined. Like most cut-and-paste functions, this one also comes in a **Transposed** variant. Note that the cell contents in this second version are the same, but their block shape is *transposed*, that is, the block definition still applies to the source matrix as before, but it is flipped along its diagonal **after** the cut is made., so its row and column dimensions are exchanged.



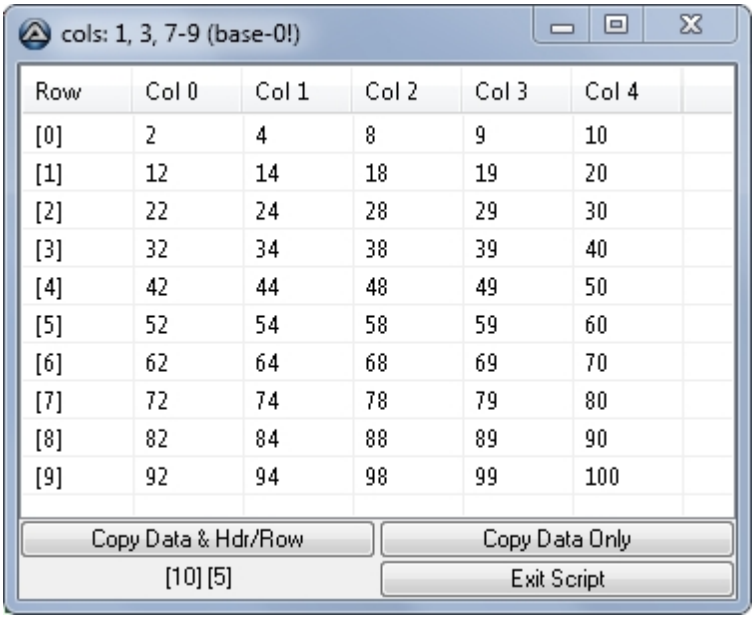

Next we want to select certain **specific** columns or rows from matrix **A**, and build a new matrix out of those columns or rows only, and maybe transpose them as well. This means we need to provide a list of our (base-zero!) row or column indices. This list is a simple string of selections separated by commas. Selections can be either single integer indices, or (inclusive) ranges of integers separated by a hyphen (dash) character, or combinations of both. The following four examples interpret this selection respectively:

- as columns
- as columns to transpose
- as rows
- as rows to transpose

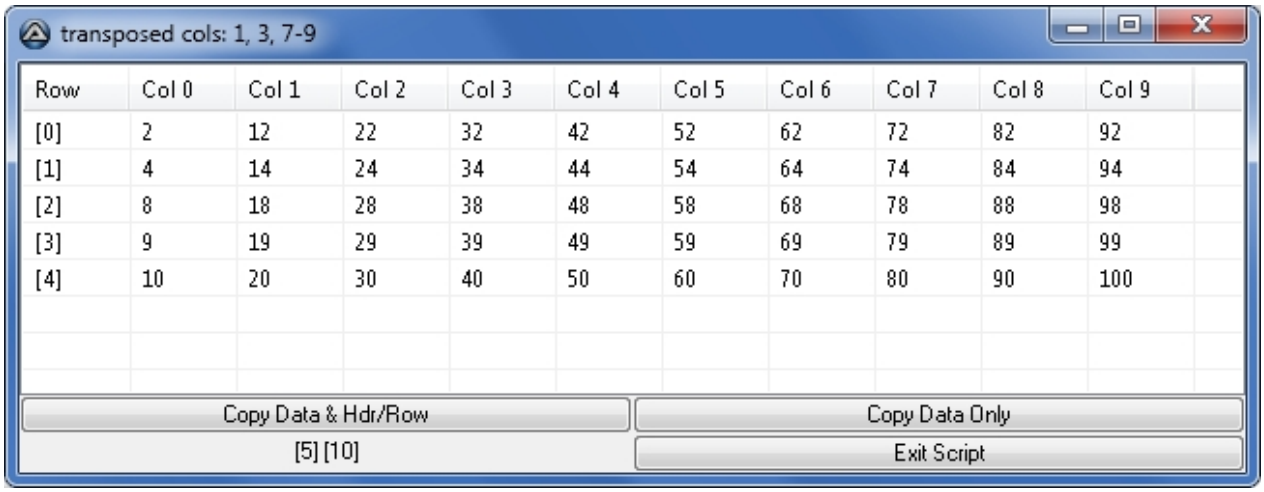
```
$selection = "1, 3, 7-9"            ; base-0

$matR = _Eigen_CreateMatrix_FromAcols ( $matA, $selection )
_MatrixDisplay ( $matR, "cols: " & $selection & " (base-0!)" )

$matR = _Eigen_CreateMatrix_FromAcols_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, "transposed cols: " & $selection )

$matR = _Eigen_CreateMatrix_FromArows ( $matA, $selection )
_MatrixDisplay ( $matR, "rows: " & $selection )

$matR = _Eigen_CreateMatrix_FromArows_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, "transposed rows: " & $selection )
```

In this code snippet, note that we keep collecting each newly-generated results matrix **R** (with a different matrix ID!) in the same variable $matR, as these matrix IDs won't be needed anymore after display. Normally, you would collect the IDs of different outputs in different variables, so you can distinguish them in subsequent usage.

Our selection string $selection defines single indices 1 and 3, plus the last three (7-9). Note that it does not specify whether these are rows or columns. The first two slicing operations interpret them as column references, whereas the last two select rows instead. The two column slicing results are shown below. Now here's a question for you: if we selected indices 1, 3, and 7 to 9 (which we did), why are the cell values in the first row shifted by one unit, that is, 2, 4, and 8 to 10?



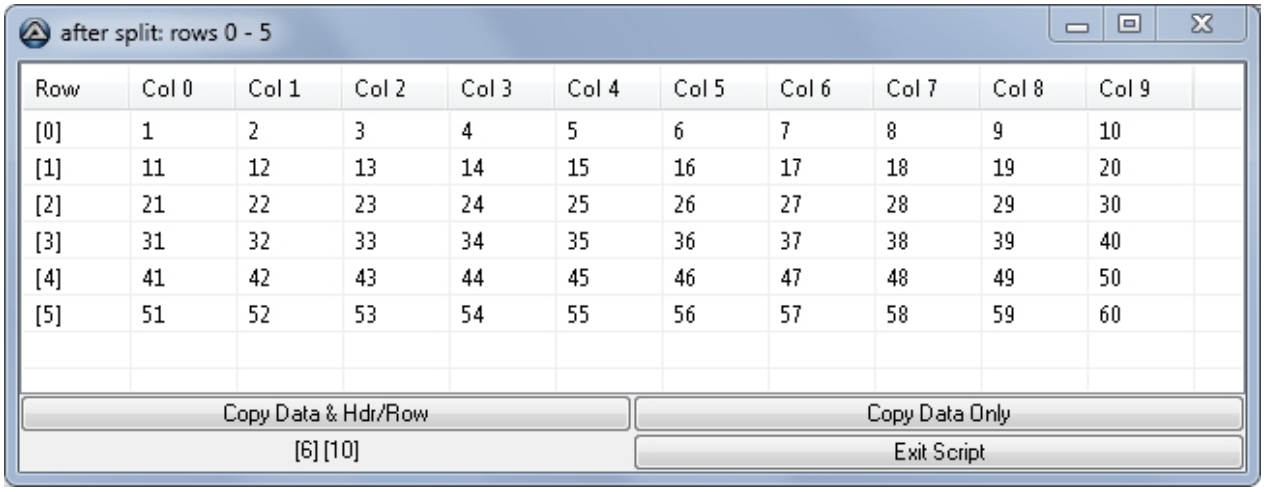

You'll find a clue to the answer in the matrixdisplay window's title up above. Below is the same matrix, but after transposition, the result of calling _Eigen_CreateMatrix_FromAcols_Transposed().

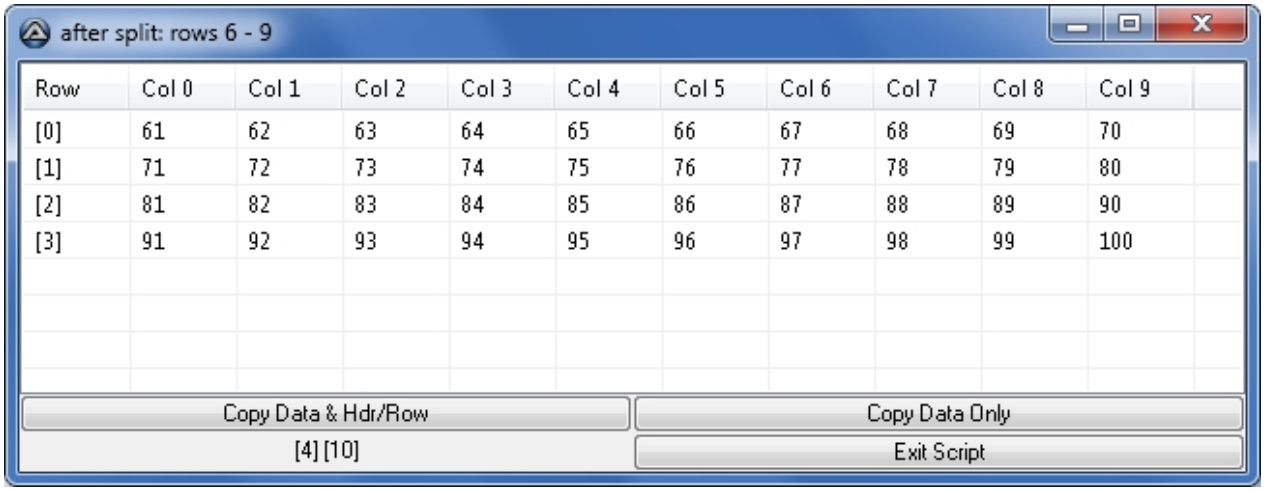



Now what if we wish to **split** a matrix into two parts, not necessarily of equal size? We could of course use the aforementioned selection functions twice, but there is a better way, using dedicated "Split" functions. Instead of providing two ranges of the indices we wish to retain, we define a single index at which the matrix is to be sliced in two. These functions thus produce **two** new matrix outputs, and the index you parse defines the **first** column or row of the **second** output. In the next few code snippets, we'll only be slicing horizontally, that is, acting on rows.

```
$slice_at_index = 6       ; the first index of the SECOND matrix
_Eigen_SplitMatrix_FromArow ( $matA, $slice_at_index )
; When you're holding a chainsaw, everything looks like a tree!

$matB = _Eigen_GetActiveMatrix ( "B" )  ; use it or lose it!
$matC = _Eigen_GetActiveMatrix ( "C" )
_MatrixDisplay ( $matB, "after split: rows 0 - " & $slice_at_row - 1 )
_MatrixDisplay ( $matC, "after split: rows " & $slice_at_row & " - " & _Eigen_GetMatrixRows ( $matA ) -
1 )
```

Something funny is going on here (funny-strange, not funny-haha): we're not collecting the results matrix ID, just calling function _Eigen_SplitMatrix_FromArow(). This is because this function produces more than a single output. It'll get much worse too; in the Regression Tutorial you'll encounter decomposition functions that can return many results simultaneously (check out this scary table if you dare). So whenever this is the case, the **Eigen4AutoIt** environment employs a more sophisticated interface, through which either you, or the function you call, declares several generic output matrices, identified by a single letter (A-Z), as **active** prior to calling the dll. Split functions expect one generic input matrix "A," and always produce two generic outputs called matrix "B" (from the topmost row c.q. leftmost column, up to the defined index) and matrix "C" (the remainder of the input matrix, from the defined index up to the end). These names are just placeholders, and will be associated with newer matrix IDs when any subsequent function refills them with other results. So it's a case of "use it or lose it;" as soon as the function returns, we have to collect those IDs and stick them in our own variables for further usage, which is what we do by calling _Eigen_GetActiveMatrix(). So whenever you use such functions, you need to figure out (by consulting the function description in this Help document) which generic letters are returned at the other end, and transfer their matrix IDs to more durable variables of your own choosing, for example, to display the two results of our row slicing action with _MatrixDisplay():





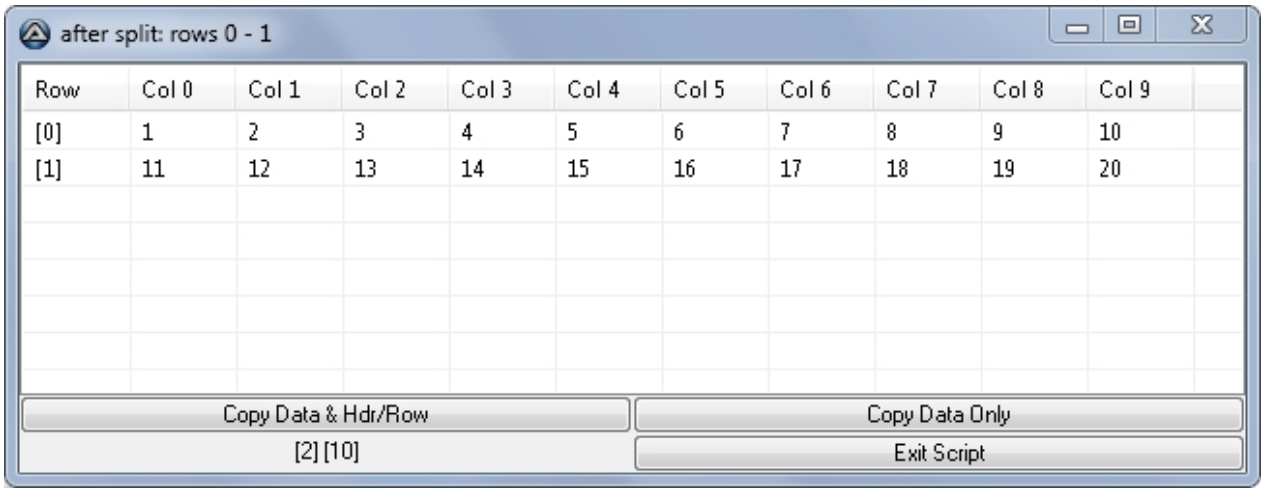

Okay, so now we already have a matrix **B** and **C**. What happens if we split matrix **B** itself again? Wouldn't this create a self-referencing conflict, as matrix **B** is now both input and output? Actually, no. Always keep in mind the distinction between:

- generic matrix placeholders such as "A," "B," "C,"
- matrix ID container variables such as $matA, $matB, $matC, and
- actual matrices such as **A**, **B**, **C** (in bold type).

In the following code snippet, we call the same split function once more, but we parse the matrix ID in variable $matB as our input matrix, and afterwards collect the two resulting IDs associated with generic outputs "B" and "C" in new variables $matBnew and $matCnew. Internally, it is the matrix IDs that provide the unique identification of our targets, not how we or the functions we call refer to them, through variables or "activation."

```
$slice_at_index = 2       ; Let's chop up that nasty output matrix B
_Eigen_SplitMatrix_FromArow ( $matB, $slice_at_index )   ; hold real still now...

$matBnew = _Eigen_GetActiveMatrix ( "B" )        ; Extra, extra! new IDs!
$matCnew = _Eigen_GetActiveMatrix ( "C" )        ; Read all about it!
_MatrixDisplay ( $matBnew, "after split: rows 0 - " & $slice_at_row - 1 )
_MatrixDisplay ( $matCnew, "after split: rows " & $slice_at_row & " - " & _Eigen_GetMatrixRows ( $matB )
- 1 )
```

The new generic results matrix "B," whose matrix ID is now stored in $matBnew, looks like this:

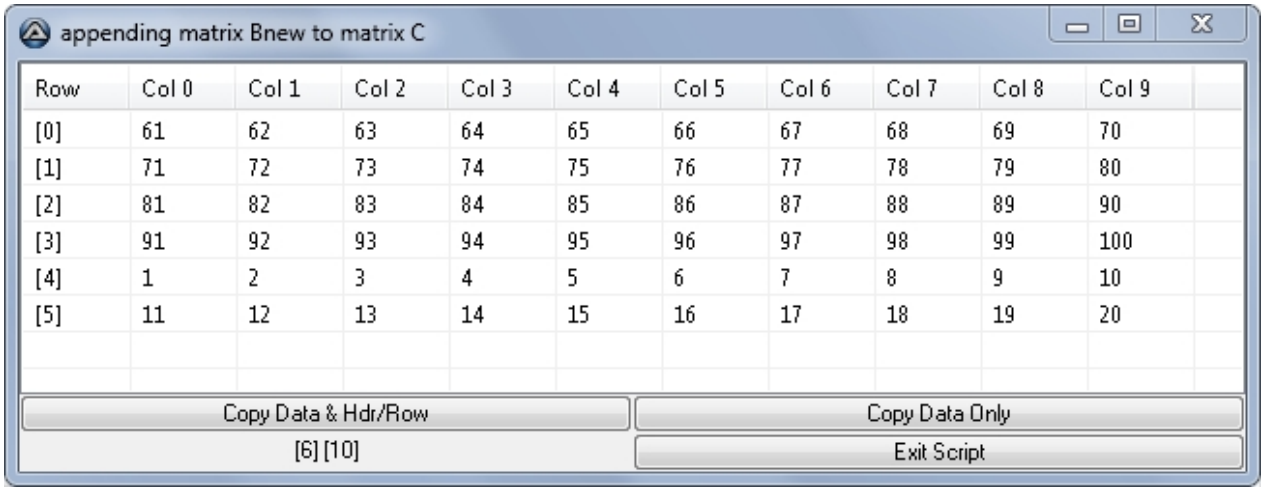


So much for cutting, but pasting is also possible, provided the **other dimension** than the one we're pasting **matches** for the two inputs. In this case, we've created several *row*-based slices, so if we wish to glue two of them together, their number of *columns* needs to match, otherwise an error will be reported. As all our examples contain remnants of the same source matrix **A**, this is of course not a problem here.

For our final example, we'll stick that last **Bnew** slice (rows 0-1 of matrix **A**) at the end of our previously-created **C** slice (rows 6-9 of of **A**). When we call the function, generic input matrix "A" is associated with the ID contained in $matC, and generic input matrix "B" is associated with the ID contained in $matBnew. A single compound result is produced, whose ID is returned directly as matrix **R**, captured (again) in variable $matR. It may take a bit of getting used to, but it's a very flexible system, with other advantages that will become apparent in later Tutorials.

Finally, ultimate **Eigen4AutoIt** function _Eigen_CleanUp() releases all matrices from memory, closes the dll, and we're done. Please drive home safely.

```
$matR = _Eigen_CreateMatrix_FromABrows ( $matC, $matBnew )
_MatrixDisplay ( $matR, "appending matrix Bnew to matrix C" )

_Eigen_CleanUp ()       ; Done and done!

MsgBox ( 0, "Eigen4AutoIt", "This concludes this Tutorial." )
```

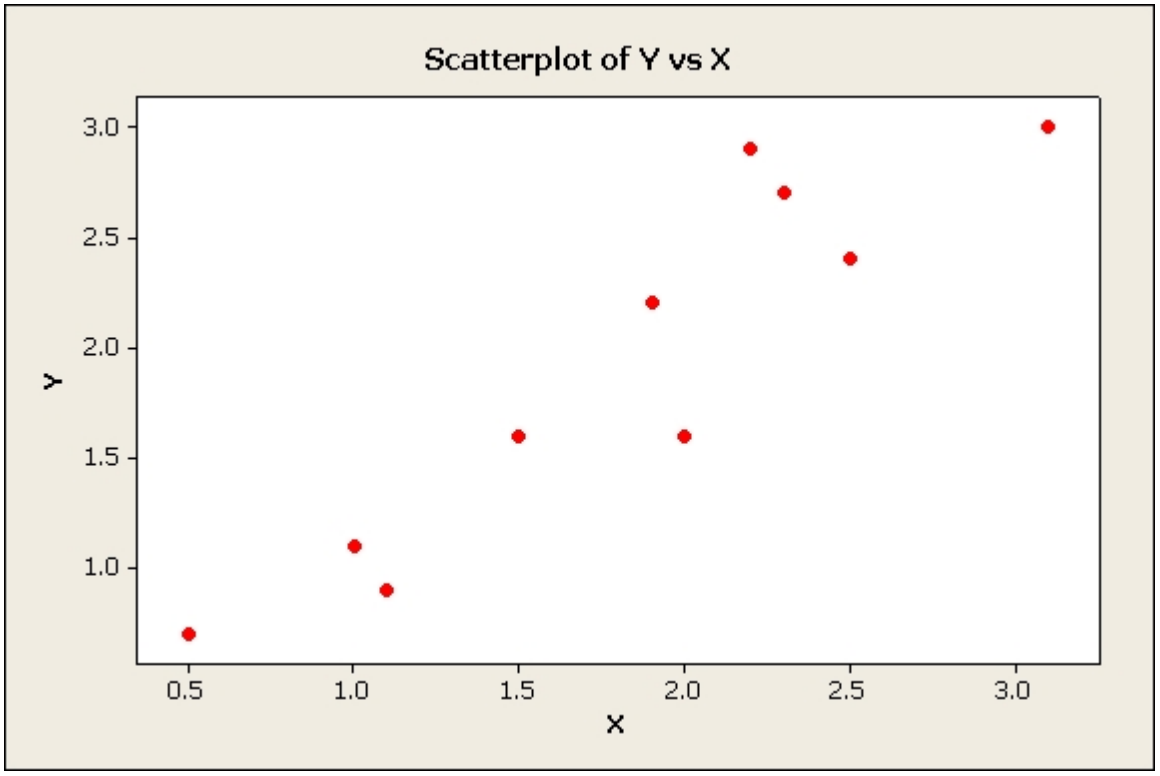





## Regression

# Tutorial - Regression

**Regression** fits a linear model to observations. This is called **linear solving**. It only hurts a little bit. The word "linear" here means that terms are simply added to produce a compound result, which can involve several different variables (y = a.X + b), or the same variable in different guises (d = a.X + b.$X^2$ + c.$X^3$.) However, it does **not** mean that the fit *itself* has to be linear (that is, a straight line); a Taylor expansion is a linear model, as is a Fourier series, Legendre polynomials, and several other fancy mathematical swear words. Any additive combination of contributing factors can be analysed and quantified. This is why much energy is spent on reformulating non-linear problems in linear terms, so that linear solvers can be brought to bear on the problem. Speaking of which, here's ours for today, a point cloud in 2D (trumpets!):




This dazzlingly beautiful graph is vertically a bit challenged (i.e., the aspect ratio is wrong), but that doesn't really matter for this Tutorial. Think of each axis as a variable, and each of the ten points as a simultaneous measurement of both variables, resulting in a pair of co-ordinates that can be plotted in 2D. When we store such data sets, the number of observations is usually much greater than the number of variables, so we like to store such pairs as long lists, with the rows representing observation events, and the columns the observables. So here we end up with ten rows and two columns of data. Let's set up a work environment and read those values in by means of an array.

```
#include "Eigen4AutoIt.au3" ; don't leave home without it

_Eigen_Startup()          ; you could use _Eigen_Startup ( "double" ) for more accurate results
_Eigen_SetListMarker()  ; to enable a cleanup afterwards

$rows = 10
```

```
$variables = 2
Local $arrayA [ $rows ][ $variables ] = [ _
        [ 2.5,  2.4 ], _
        [ 0.5,  0.7 ], _
        [ 2.2,  2.9 ], _
        [ 1.9,  2.2 ], _
        [ 3.1,  3.0 ], _
        [ 2.3,  2.7 ], _
        [ 2.0,  1.6 ], _
        [ 1.0,  1.1 ], _
        [ 1.5,  1.6 ], _
        [ 1.1,  0.9 ] ]
_ArrayDisplay ( $arrayA, "original data" )          ; show data array A

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )      ; data matrix A
```

Large datasets we would of course read in directly from a file, using _Eigen_LoadMatrix(), or we would import them from other file formats first (ASCII delimited, Excel Workbooks, Xbase databases) using the MatrixFileConverter utility.

It's fairly obvious what we're doing in the above code snippet: we create an array, stick the data in, display it, and create a matrix from that array. Since we didn't explicitly tell that last function to ***not*** copy the data, all values are copied into the matrix cells too. Life can be simple sometimes.

Let's assume that the two columns represent different aspects of the measurements; Column **0** (matrix indexing is always base-0!) we'll consider to be the independent part, which we, the observers controlled, whereas Column **1** is the dependent response. Note that in real life, both parts can have multiple columns; this is just the simplest possible scenario.

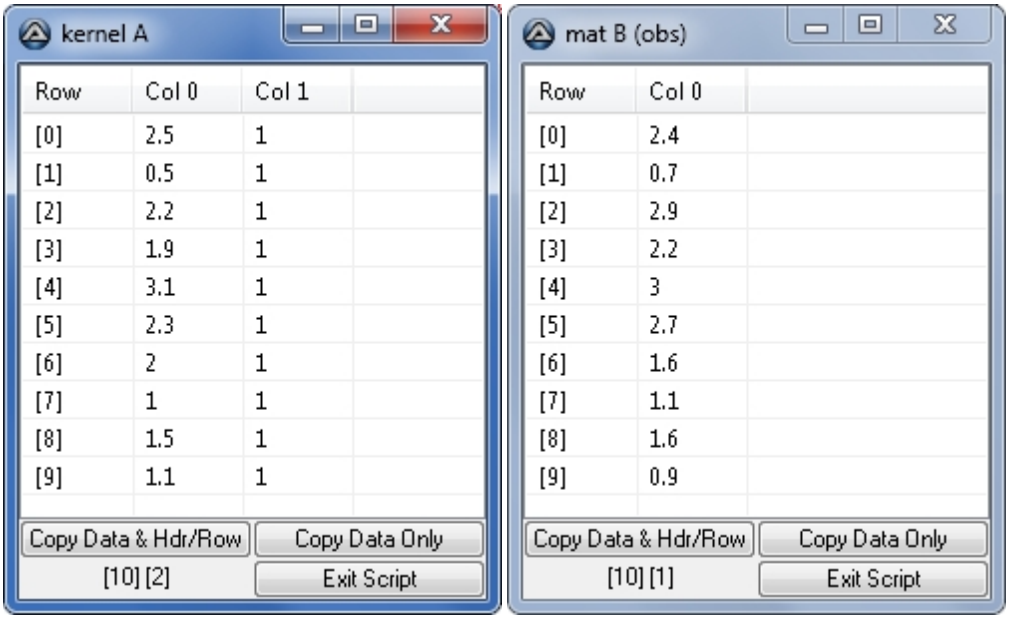
Suppose it is incredibly important that we can predict **B**'s value reasonably well, based on whatever **A** might be in future. In fact, the fate of the entire world hangs in the balance, and YOU have to come up with the answers, and quickly too! Well, don't just sit there! Start saving the world!

We need to come up with some kind of model that captures the relationship between the independent and dependent parts of the data. Then we can feed that model new independent values, and see what it predicts as dependent result for those new inputs. More specifically, we wish to apply linear solving, and we'll start with the grand-daddy of them all, **Least-Squares** (invented in the early 19th century by Legendre and Gauss).

For this to work we will need to split our original data into two separate matrices, one for the independent part, and one for the dependent part. We'll try a simple line fit first, for which we need two columns: one for the slope (the line's angle with the horizontal) and one for the intercept (the point where it intersects the dependent variable's axis, that is, where the independent variable is zero). We'll use matrix **A**'s column 1 as placeholder for the constant ***c*** (the intercept) in the fit: a.X + 1.**c** = y (note the constant's multiplication by the value one):

```
$matB = _Eigen_CreateMatrix ( $rows, 1 )          ; create a 1-column data buffer (a column vector) for
the Y response variable

_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 ) ; first source  (col 1), then destination (col 0)

_Eigen_SetOnes_Col ( $matA, 1 )           ; the "one"-multiplier to compute the intercept constant in our
line fit

_MatrixDisplay ( $matA, "kernel A" )      ; now matrix A looks like this...
```

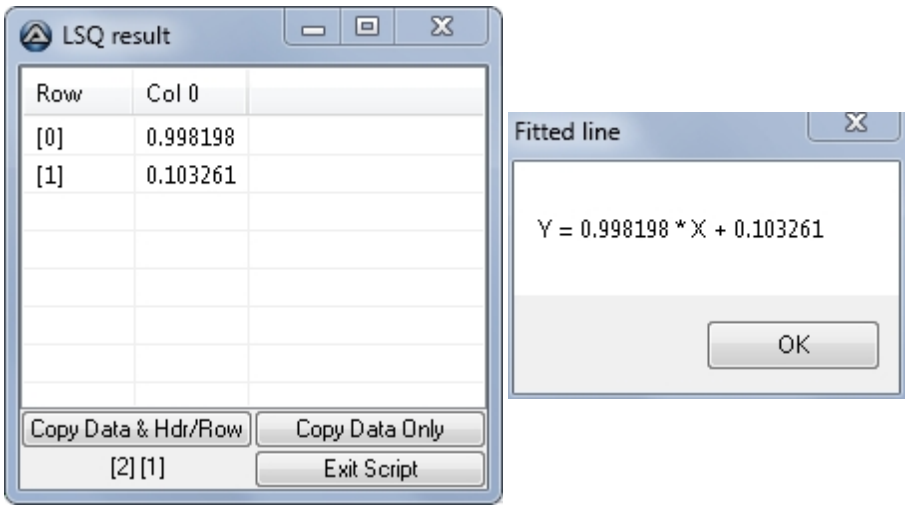
```
_MatrixDisplay ( $matB, "mat B (obs)" )  ; ...and our observations are now a separate column vector
```




Now we'll compute the least-squares linear fit (which assumes Gaussian-distributed errors), obtaining matrix **X** (whose number of rows ("rowsX") equal "colsA" (the number of columns of matrix **A**), and whose number of columns ("colsX") equal "colsB," the number of columns of **B**). In this case, rowsX = 2 and colsX = 1, so **X** is actually a *Colvector* of size 2 here. The first value, at cell coordinates (0,0) is the slope; the second cell contains the intercept.

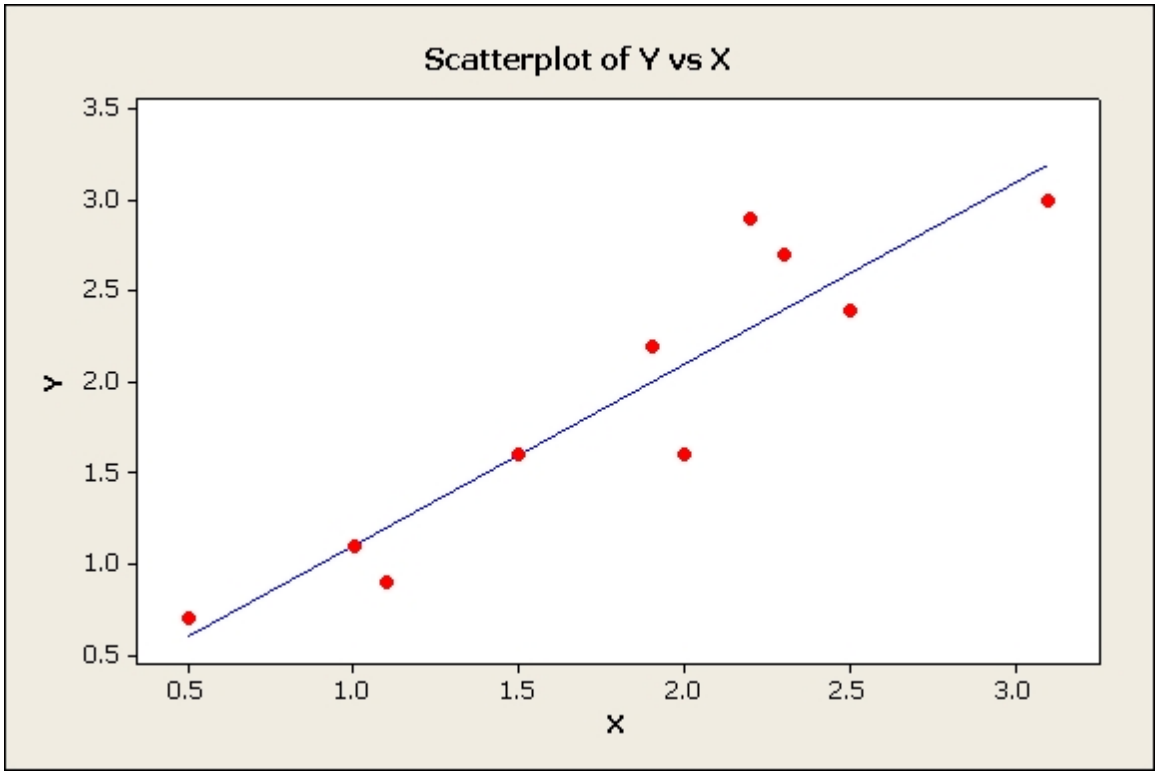
```
$matX = _Eigen_LeastSquares ( $matA, $matB )  ; power up the model-fitting engine

_MatrixDisplay ( $matX, "LSQ result" )    ; returned: two coefficients: slope and intercept; that'll be five bitcoins, please!

$originalFit = "Y = " & _Eigen_ReadMatrixValue ( $matX, 0, 0) & " * X + " & _Eigen_ReadMatrixValue ( $matX, 1, 0 )

MsgBox ( 0,"Fitted line", $originalFit )
```

Variable $originalFit in the snippet is filled with a composite string that includes the two line coefficients retrieved from each cell of matrix **X** with **_Eigen_ReadMatrixValue()**.



The Least-Squares result in matrix **X** contains only a single column of coefficients because we supplied the function with a single column of observations to fit (matrix **B**). But just imagine for a moment matrix **B** having twenty columns, or twenty thousand, or twenty million. In that case, results matrix **X** would contain an equal number of ***separate fits***, computed simultaneously! Most linear solvers provided in **Eigen4AutoIt** have this parallel capacity, which is a perfect example of the power of matrix computing. The size of your problem no longer matters; if it's huge. it just takes a bit longer to compute.

Okay, we obtained the description of a line through all points, which looks like this:



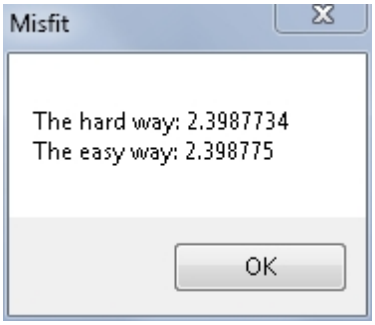

That looks like a decent fit. But is it good enough to Save The World? How much do the points actually scatter around our model estimate? We can quickly whip up a few lines of code that compute the so-called Misfit (the sum of Abs( prediction minus observation) for all observations) , like so:

```
_Eigen_CreateArray_FromMatrix ( $matA, $arrayN )        ; kernel now also stored in array N
_Eigen_CreateArray_FromMatrix ( $matB, $arrayB )        ; observations now also stored in array B
_Eigen_CreateArray_FromMatrix ( $matX, $arrayX )        ; model coefficients now also stored in array X

$summedResiduals = 0          ; running total in the next loop

For $rc = 0 To $rows - 1
        $predicted = ( $arrayX [0] * $arrayN [$rc][0] ) + $arrayX [1]          ; a.X + c = predicted
dependent value
        $summedResiduals += Abs ( $predicted - $arrayB [$rc] )          ; add the absolute difference
between prediction and observation
Next
```

On second thought, that was a really, really dumb way of doing it (duplicating containers, copying all data, and looping in AutoIt), since we can do it much faster with an indiginous **Eigen4AutoIt** function call:

```
_$summedResidualsLinear = _Eigen_Misfit ( $matA, $matB, $matX )         ; That's all, folks!

MsgBox ( 0, "Misfit", "The hard way: " & $summedResiduals & @CR & "The easy way: " &
$summedResidualsLinear )
```

There can be a slight difference between the two due to differences in rounding. Several alternatives to the Misfit (R-squared, Relative Error, and SSR) are also available. You'll find them in the Advanced Statistics section.




We're not done yet. What if some observations are more reliable than others? Well, then we can **weight** each observation with a square weighting matrix **W** to express this difference. In the next code snippet, we'll first weight with an *Identity matrix*, the scalar equivalent of unity. This weights each observation equally with 1, so we retrieve the same result as before, just as a test. Subsequently, we'll give the fifth observation (the top-rightmost point) a much higher weight and re-fit, which should slightly alter the line coefficients in the latest model.

```
$matW = _Eigen_CreateMatrix ( $rows, $rows )              ; square weighting matrix W
_Eigen_SetIdentity ( $matW )     ; we could also have used _Eigen_CreateMatrix_Identity() directly
_MatrixDisplay ( $matW, "flat weighting matrix (square: rows x rows)" )

$matX = _Eigen_LeastSquares ( $matA, $matB, $matW ) ; compute a new model fit
_MatrixDisplay ( $matX, "LSQ flat-weighted" )     ; should be the same result, because all weights were
equal

_Eigen_WriteMatrixValue ( $matW, 4, 4, 10 )       ; coordinates = (4,4), because cells are indexed base-0!

$matX = _Eigen_LeastSquares ( $matA, $matB, $matW ) ; here we go again...
_MatrixDisplay ( $matX, "LSQ Reweighted" )        ; hold the phone; this one's different!

$weightedFit = "Y = " & _Eigen_ReadMatrixValue ( $matX, 0, 0 ) & " * X + " & _Eigen_ReadMatrixValue
( $matX, 1, 0 )

MsgBox ( 0, "Fitted line", "unweighted: " & $originalFit & @CR & "weighted:      " & $weightedFit )
```

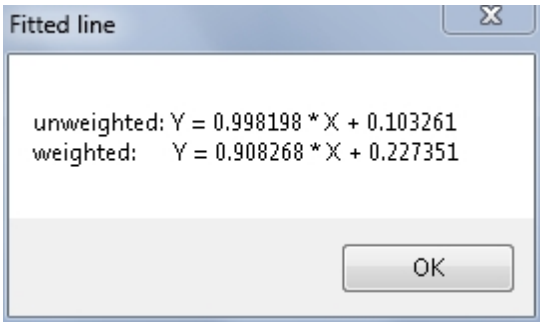
NEW weighting matrix (note cell 4,4's changed content!)

| Row | Col 0 | Col 1 | Col 2 | Col 3 | Col 4 | Col 5 | Col 6 | Col 7 | Col 8 | Col 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| [0] | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| [1] | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| [2] | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| [3] | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| [4] | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 |
| [5] | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| [6] | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| [7] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| [8] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| [9] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

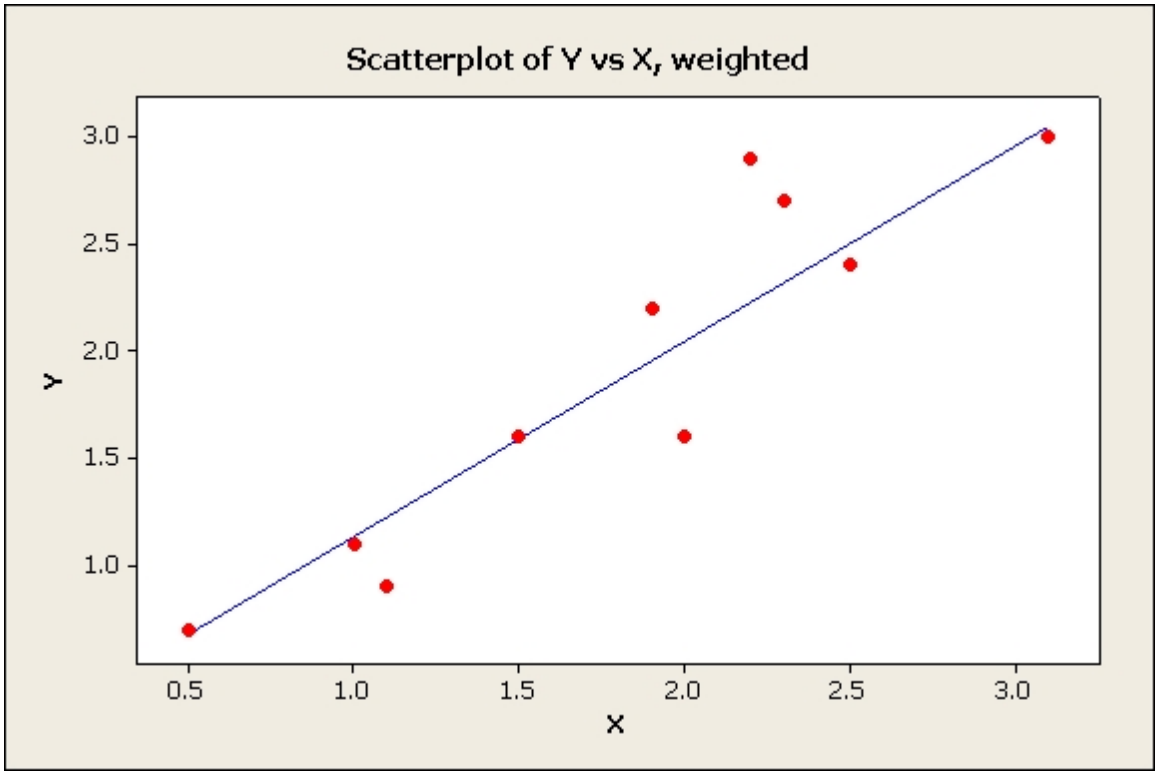
Copy Data & Hdr/Row | Copy Data Only

[10] [10] | Exit Script

In the last fit, both coefficients have changed, to accommodate the higher weight given to the fifth observation (which is the one farthest from the origin):



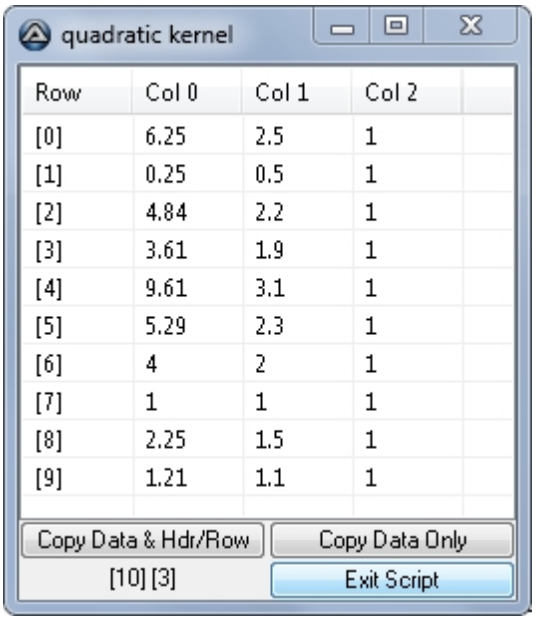

and if you compare the new plot with the original one, you'll notice that the new line now almost touches both the lowest and the highest point; the latter is the one whose weight we just increased tenfold.



Straight lines are easy. Let's fit a **quadratic curve** now ( $a.X^2 + b.X + 1.c = y$ ) to our data. For this we need to build a new kernel matrix, adding an extra column for the squared parameter. Of course we can re-use the contents of the original kernel; we simply copy the two columns across as a single **block**. Please observe:

```
$matN = _Eigen_CreateMatrix ( 10, 3 )  ; create a new container

$startRow_src = 0       ; define the block at source: top left, source row = 0
$startCol_src = 0       ; top left, source column = 0
$blockRows = $rows      ; use all rows for our block
$blockCols = 2          ; our block is 2 columns wide
$startRow_dst = 0       ; keep block in top row at destination
$startCol_dst = 1       ; destination column = 1 (start block at col1 in destination)

_Eigen_Copy_Ablock_ToBblock ( $matA, $matN, $startRow_src, $startCol_src, $blockRows, $blockCols,
$startRow_dst, $startCol_dst )
```

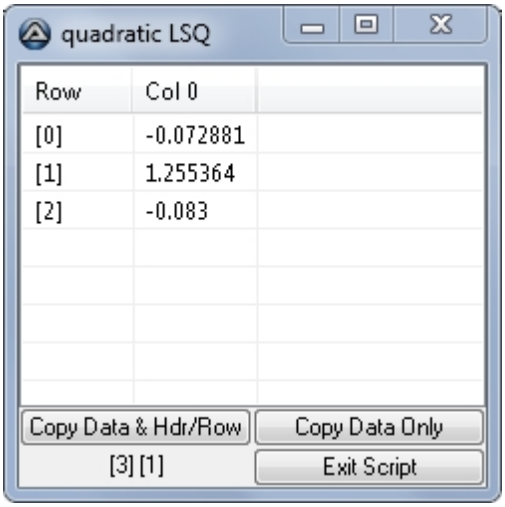
Afterwards, we copy column 1 to column 0 internally (see below). Note that we parse the same matrix ID twice here ($matN); not all **Eigen4AutoIt** functions allow this. Subsequently, we replace the contents of column 0 with their value squared. We wish to act on each cell individually, so we use a Cwise ("cell-wise") function, and since we only wish to change one column, we call a Cwise*_Col function. Then we produce a quadratic model fit, and evaluate how well it performs.

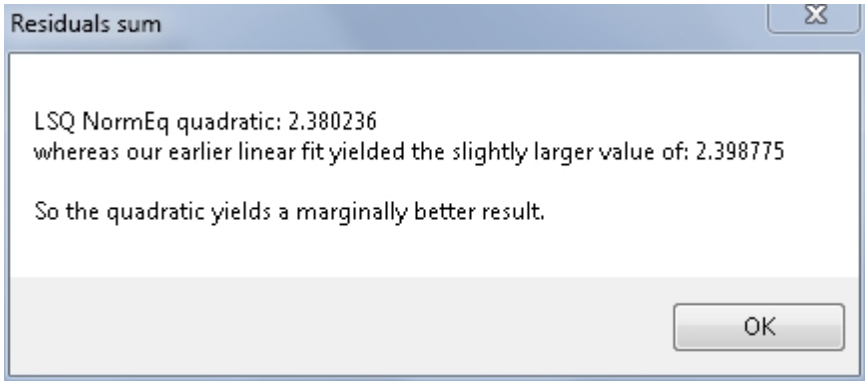


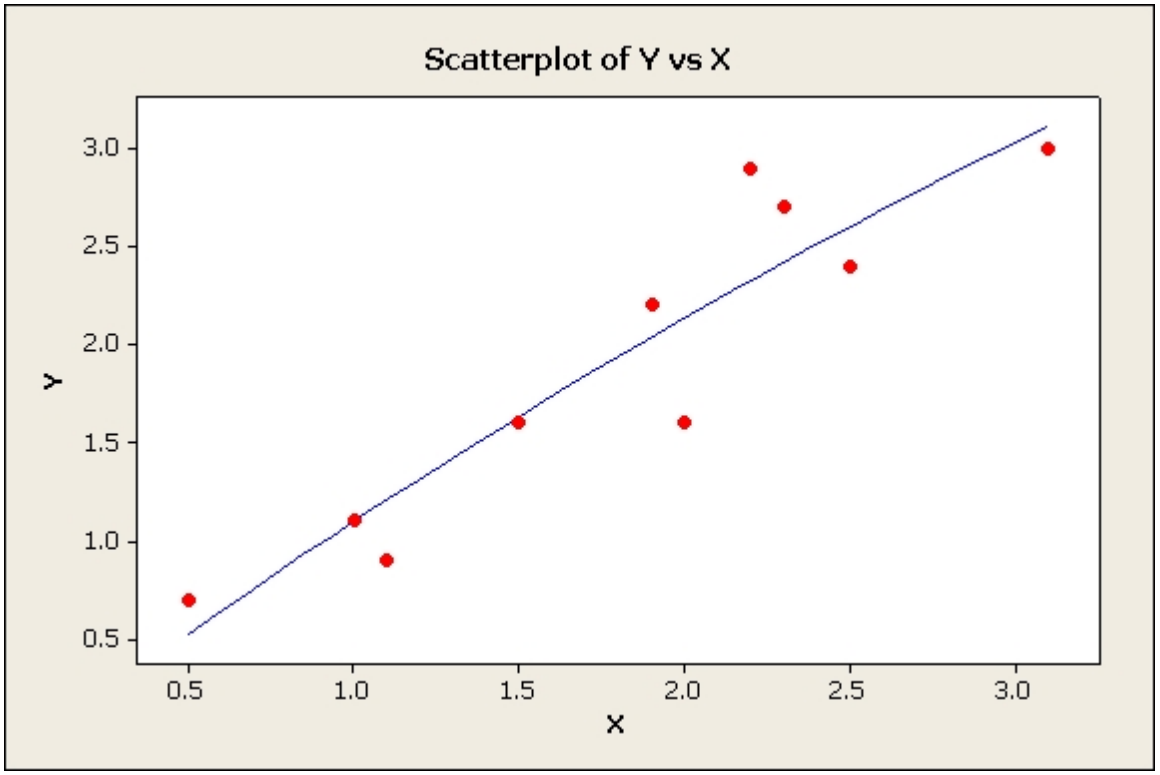

```
_Eigen_Copy_Acol_ToBcol ( $matN, $matN, 1, 0 ); source = column 1, destination = column 0

_Eigen_CwiseUnaryOp_Col_InPlace ( $matN, 0, "square" ); Cwise functions accept operator strings

_MatrixDisplay ( $matN, "quadratic kernel" )      ; 3 columns, to compute 3 model coefficients
```

```
$matX = _Eigen_LeastSquares ( $matN, $matB ) ; no longer using the weighting matrix

_MatrixDisplay ( $matX, "quadratic LSQ" )         ; The best-fitting parameters abc for our (a.x^2 + b.x +
1.c = y)
```

```
$summedResidualsQuadratic = _Eigen_Misfit ( $matN, $matB, $matX )     ; good, bester, besterer, bestest?

MsgBox ( 0, "Residuals sum", "LeastSquares quadratic: " & $summedResidualsQuadratic & @CR & _
"whereas our earlier linear fit yielded the slightly larger value of: " & $summedResidualsLinear & @CR & @CR & _
"So the quadratic yields a marginally better result." )
```






Next, we will compare Gaussian Least Squares to several linear solvers provided in the main decompositions. These are heavy-duty number crunchers that can produce multiple outputs that break up the original inputs into various other matrices with desirable properties. To engage with these, we will be using a more sophisticated interface that keeps track of which potential inputs and outputs are to be considered "**active**" in the current application. How does this work?

Generic matrix **X** will receive the model fit if we define it as "active" before calling the decomposition. We'll do this once, and then *keep* it active in all subsequent calls to various other decompositions (simply by not resetting it to inactive). Unlike generic input matrices **A** (always the kernel) and **B** (always the observations), we do not parse the matrix ID of matrix **X** to the function directly, as this would tell the decomposition that we are providing an existing container to hold the new result. Although this re-use would require less memory, you'd need to make sure that your provided container *exactly* fits the expected output dimensions. Instead, we'll flag it as "active" without providing a matrix ID, which tells the decomposition (in this case: JacobiSVD) to create a new output matrix for it, which will automatically have the correct dimensions. However, this does imply that we'll need to "collect" the matrix ID of that new container (think of it as your matrix's "handle") after the decomposition is done. It all sounds far more complicated than it really is:

```
_Eigen_SetActiveMatrix ( "X", True )       ; activate output matrix X only; True = clear any other active flags first

$enableSpecs = False  ; set some operational switches...
$fullPivoting = False     ; too much work here (yawn)
$colPivoting = True       ; default, medium effort, quite stable (trust me on this)
$computeThin  = True  ; don't waste your CPU breath; Think Thin!

_Eigen_Decomp_JacobiSVD ( $matN, $enableSpecs, $fullPivoting, $colPivoting, $computeThin, $matB ) ; The Decomposition Call
If @error Then Exit       ; always a good idea to check after a major computing event
```

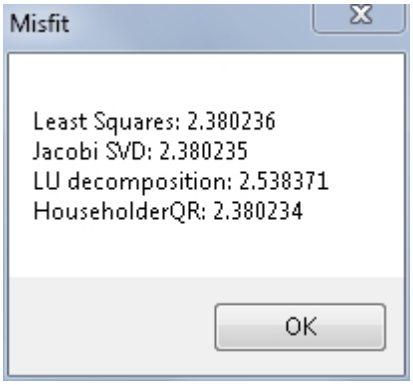
```
$matX = _Eigen_GetActiveMatrix ( "X" ) ; here we collect the matrix ID of generic output matrix X
_MatrixDisplay ( $matX, "Jacobi SVD result" )     ; our new model coefficients

$summedResidualsSVD = _Eigen_Misfit ( $matN, $matB, $matX ) ; we might as well, now that we know
how
```

As the last three lines show, any output we declared active before the decomposition call can be collected *individually* after the call with _Eigen_GetActiveMatrix(). Once we have stuck its matrix ID in a variable, we can use it however we like, whereas its "active slot" can then safely be overwritten with a different matrix ID in the next call. As explained before, we do not de-activate matrix **X**, which means that the next two decompositions (LowerUpper and HouseholderQR) will also return this output, generating **new** matrix IDs:

```
$fullPivoting = False     ; set another operational flag ($enableSpecs is still False)

_Eigen_Decomp_LowerUpper ( $matN, $enableSpecs, $fullPivoting, $matB )          ; crunch time!
If @error Then Exit

$matX = _Eigen_GetActiveMatrix ( "X" ) ; Quick, grab it, before it is overwritten again!
MatrixDisplay ( $matX, "LU decomposition result" )        ; gotcha!

$summedResidualsLU = _Eigen_Misfit ( $matN, $matB, $matX )   ; compute residuals
```

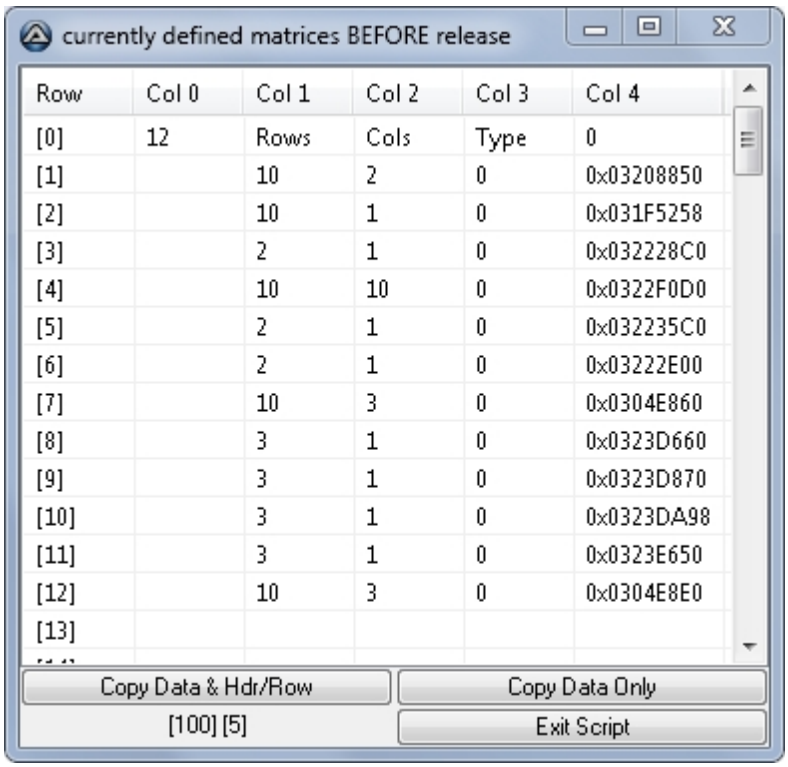
```
$fullPivoting = True      ; set several operational flags ($enableSpecs is still False)
$colPivoting = False      ; $computeThin is still True; keep it that way

_Eigen_Decomp_HouseholderQR ( $matN, $enableSpecs, $fullPivoting, $colPivoting, $computeThin,
$matB ) ; munch munch, gobble, gobble...
If @error Then Exit

$matX = _Eigen_GetActiveMatrix ( "X" )
_MatrixDisplay ( $matX, "HouseholderQR result" )

$summedResidualsQR = _Eigen_Misfit ( $matN, $matB, $matX )  ; compute residuals

MsgBox ( 0, "Misfit", _              ; and the winner is: ...
"LSQ NormEq: " & $summedResidualsQuadratic & @CR & _
"Jacobi SVD: " & $summedResidualsSVD & @CR & _
"LowerUpper: " & $summedResidualsLU  & @CR & _
"HouseholderQR: " & $summedResidualsQR  & @CR )
```

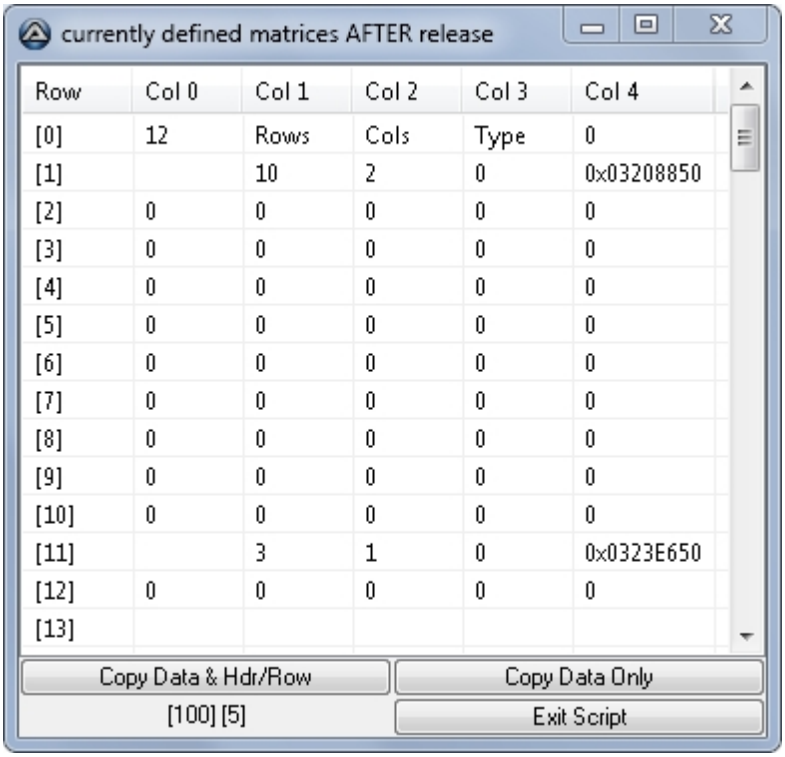
Each decomposition has its own input and settings requirements, and some take much longer than others, and/or require more memory. In some cases, a solution may be invalid, or is such a poor fit that it has to be rejected; please **always** check that your results are reasonable. Here **LowerUpper** (LU) decomposition does a worse job than all other procedures.




We're just about done, but we've used quite a few resources, and these remain allocated. Each time we obtained a model fit **X**, a new matrix was created, as you can see in the global $MatrixList array; every entry with non-zero dimensions (rows, cols) still exists. But remember that we placed a ListMarker at the start of this tutorial? Now we can finally use it to release all matrices created thereafter, with the exception of the original data (matrix **A**) and the final result (the last matrix **X**) in this case.

```
_ArrayDisplay ( $MatrixList, "currently defined matrices BEFORE release" )

_Eigen_ReleaseFromMarker_Except ( $matA, $matX )      ; you can retain up to ten matrices per call

_ArrayDisplay ( $MatrixList, "currently defined matrices AFTER release" )  ; note the new zero entries

_Eigen_Cleanup ()        ; this clears the enitre MatrixList, and resizes it to zero entries

MsgBox ( 0, "Eigen4AutoIt", "This concludes this Tutorial." )
```






Congratulations! You have completed the Regression Tutorial.



### Principal Components Analysis

# Tutorial - Principal Components Analysis

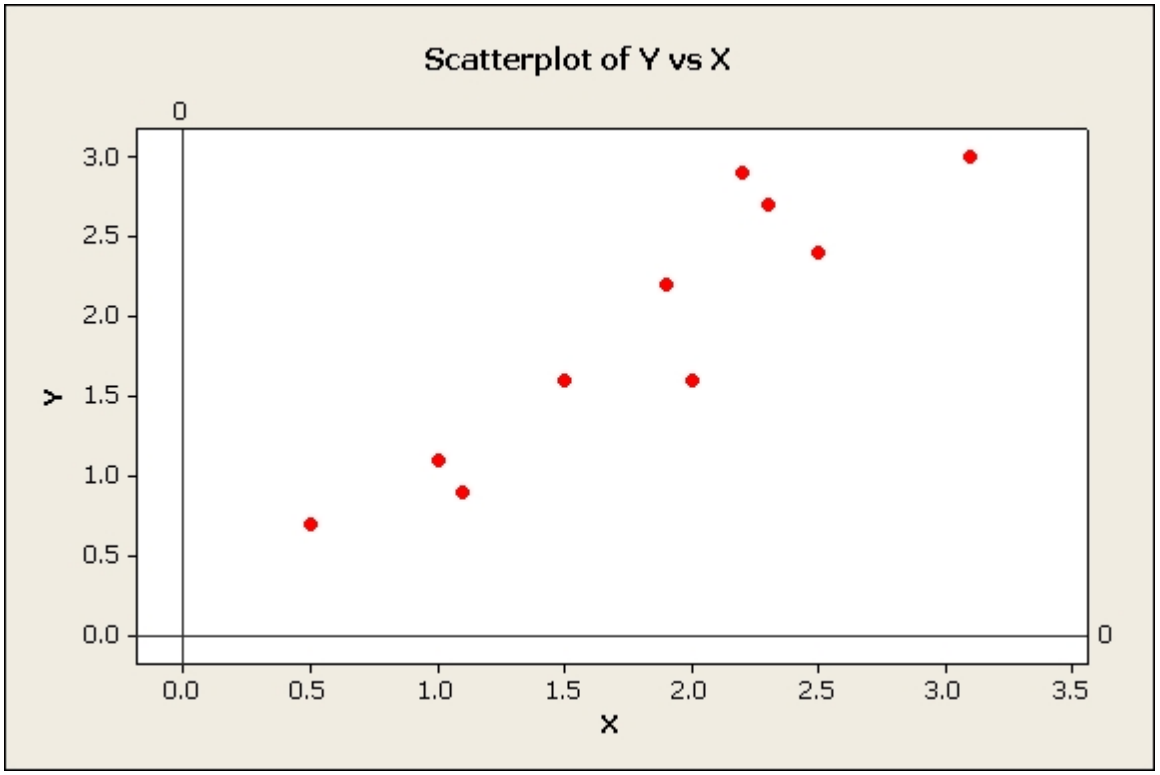
This intermediate-level tutorial illustrates how to combine several simple building blocks to construct an advanced procedure called Principal Components Analysis, a powerful statistical technique used in pattern recognition (hereafter denoted "PCA"). It is assumed that you are already familiar with creating and running **Eigen4AutoIt** scripts, and are ready for the big time!

This simplified case-study is based on this resource by L.I. Smith.; you may also find these two pdfs helpful. Alternatively, you could google "PCA tutorial" or similar terms for additional explanation. The argument as presented here goes like this:

1. we can project data points into "principal component space" in which independent axes (dimensions) are ordered by how much data variability they map (largest to smallest);
2. this operation is reversible (back-projection);
3. we can ignore any number of higher-order principal component axes prior to back-projection, keeping only the ones that explain most data variability (lossy compression).

Okay, let's dance! Here we have ten points in 2D space; these could be coordinates X and Y, or simultaneous observations of two physical properties, for example. If you've been a diligent observer, they may look strangely familiar...

```
#include "Eigen4AutoIt.au3" ; get ready to rumble!

_Eigen_Startup ()          ; you could use _Eigen_Startup ( "double" ) for more accurate results
_Eigen_SetListMarker () ; to enable a cleanup afterwards

$rows = 10
$cols = 2
Local $arrayA [ $rows ][ $cols ] = [ _
        [ 2.5,  2.4 ], _
        [ 0.5,  0.7 ], _
        [ 2.2,  2.9 ], _
        [ 1.9,  2.2 ], _
        [ 3.1,  3.0 ], _
        [ 2.3,  2.7 ], _
        [ 2.0,  1.6 ], _
        [ 1.0,  1.1 ], _
        [ 1.5,  1.6 ], _
        [ 1.1,  0.9 ] ]
$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )       ; data matrix A

MatrixDisplay ( $matA, "original data" )
```

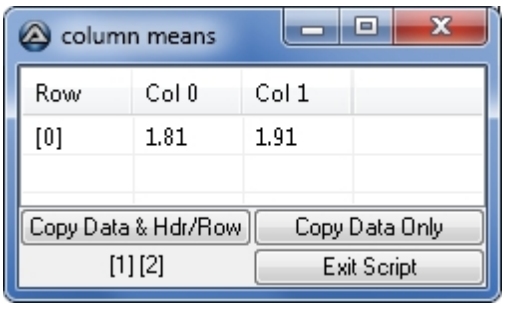



When we plot these data, the *origin* is the intersection of the two zero axes at bottom left. The dots are not centered around this origin, so first we'll translate them back there. To do this, we take the mean per column, and subtract those means from all individual values within that column. Since we'll need the means again later, we'll store those too.

The mean happens to be defined as index 3 in the list of matrix specs we can retrieve with the

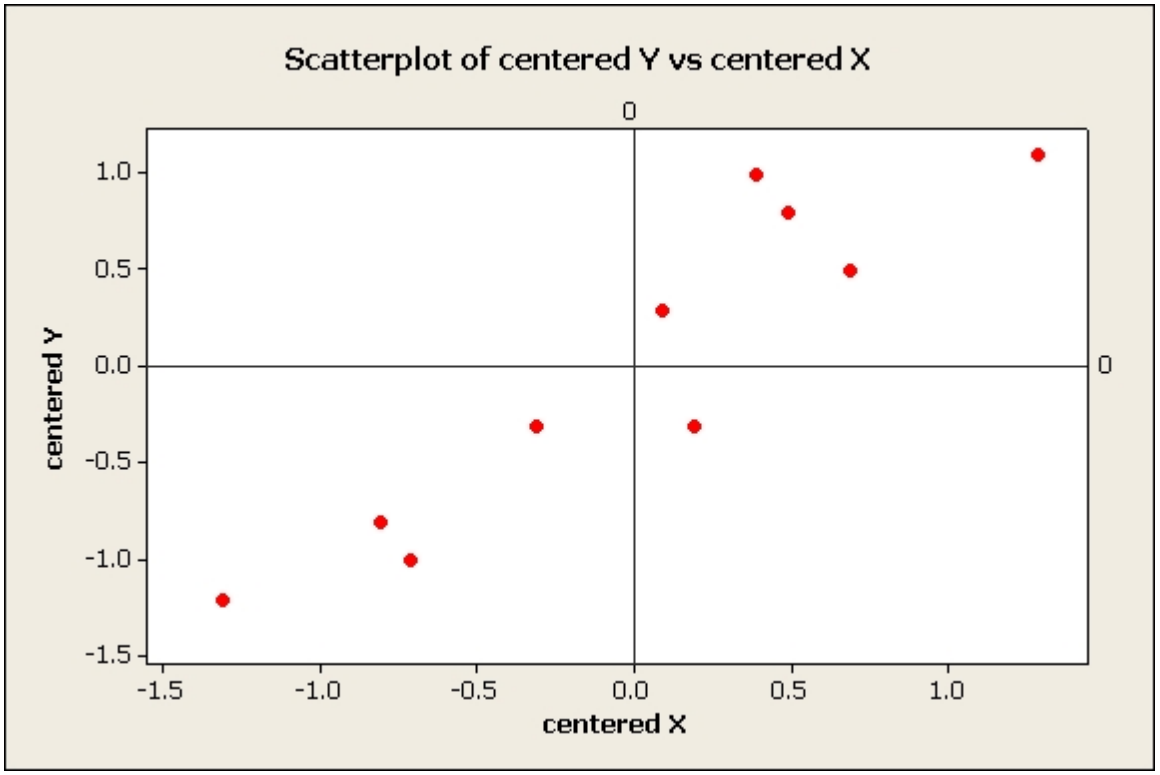
"Matrixspecs* functions. Here we use _Eigen_MatrixSpecs_Colwise_Single(); "Colwise" = repeat for each column, "Single" = retrieve a single property of the target matrix. We end up with a Rowvector **M** (a matrix with one row and multiple columns) in which each cell contains the mean for its column in matrix **A**:

```
$specID = 3     ; we wish to retrieve the mean (average)

$matM = _Eigen_MatrixSpecs_Colwise_Single ( $matA, $specID ) ; get this spec per column
_MatrixDisplay ( $matM, "column means" )        ; Rowvector M
```



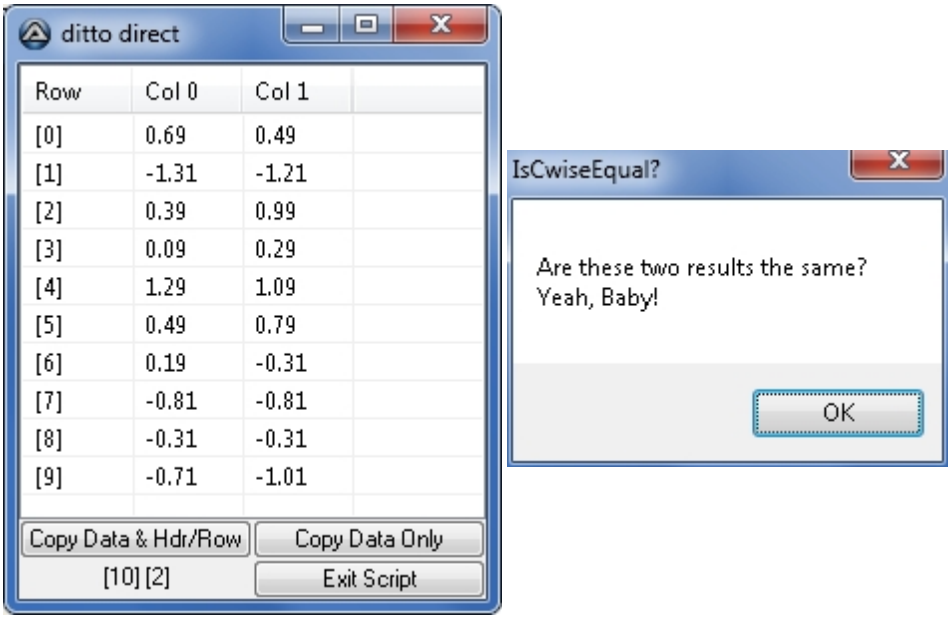

Next, we have to subtract this vector of column means from the cells in each row of **A**. We also want to keep our original data, so we first clone matrix **A**. Some explanation of terms again: "Cwise" = cell-wise, or coefficient-wise = act on each cell individually; "Rowwise" = perform the action on each row in turn; "BinaryOp" = Binary Operator = use an operator with the data from two different matrices, storing the result in the first input matrix; "InPlace" = store the result in the input matrix itself.

```
$matB = _Eigen_CloneMatrix ( $matA )  ; create a new matrix (B), duplicating matrix A

_Eigen_CwiseBinaryOp_Rowwise_InPlace ( $matB, "-", $matM )     ; NB alters content of first matrix
parsed!
_MatrixDisplay ( $matB, "Colwise mean-centered data" )
```



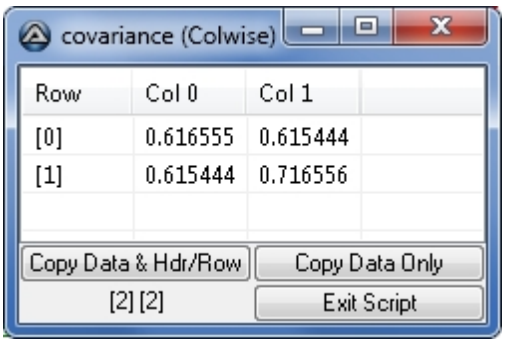

Note: if we didn't need to store those column means for later, we could have simply done this instead:

```
$matZ = _Eigen_Center_Colwise ( $matA )          ; subtract column mean from each cell in that column, for each column in turn
_MatrixDisplay ( $matZ, "ditto direct" )

MsgBox ( 0, "IsCwiseEqual?", "Are these two results the same?" & @CR & _
        ( ( _Eigen_IsCwiseEqual_AB ( $matB, $matZ, 1.0e-3 ) ) ? "Yeah, Baby!" : "No way, Bro!" ) )      ; AutoIt's ternary operator
```




PCA is all about quantifying the **dimensionality** of data variability. For starters, we need to know how broad the variability is for each variable, and how changes in variable X affect variable Y. This is called covariance. The square covariance matrix contains the data variances on the diagonal (column = variable), and the co-variance of XY pairs in the off-diagonal cells (values mirror across the diagonal, because pair XY = pair YX).

```
$matC = _Eigen_Covariance_Colwise ( $matB )
_MatrixDisplay ( $matC, "covariance (Colwise)" )
```

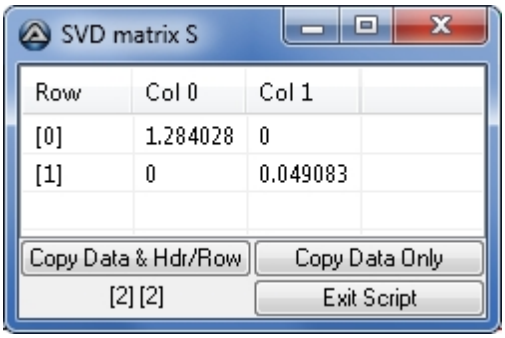



Now we bring out the big guns: Jacobi Singular Value Decomposition (or "SVD," for Singular Value Decomposition; see the Jacobi Test script for info on what else it can produce). Before summoning this fearless number cruncher, we need to tell it which outputs to produce (matrices **S** and **U** in this case, to be explained below):

```
_Eigen_SetActiveMatrix ( "SU", True )      ; True = reset all activation bit flags first
```

Next, we adjust the operational specifics of this decomposition with various flags, call the decomposition, collect the matrix ID of the returned eigen**values** (matrix **S**) and stick the latter in new vector **D**:

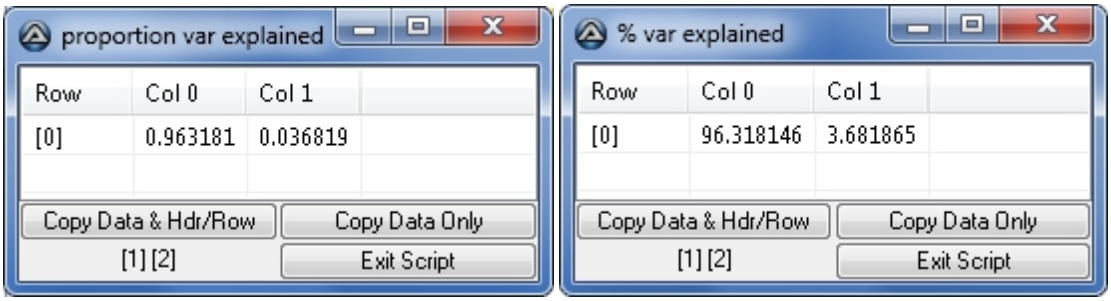
```
$enableSpecs = False  ; various scalar values describing the results; not interesting here.
$fullPivoting = False    ; Pivoting is irrelevant here, because it deals with non-square inputs,
$colPivoting =False      ;   and a covariance matrix is square by definition
$computeThin =False   ; likewise irrelevant, for the same reason

; Houston, we are GO for launch!
_Eigen_Decomp_JacobiSVD ( $matC, $enableSpecs, $fullPivoting, $colPivoting, $computeThin )
If @error Then Exit      ; if @error = 0: we're good, people!


$matS = _Eigen_GetActiveMatrix ( "S" )          ; Get the newly created matrix ID
_MatrixDisplay ( $matS, "SVD matrix S" )        ; Display the eigenVALUES = the length of each PC axis

$matD = _Eigen_CreateMatrix ( 1, $cols )        ; create a Rowvector of length $cols for it

_Eigen_Copy_Adiag_ToBvector ( $matS, $matD ) ; store the eigenvalues here
```




The eigenvalues are size-ordered, and quantify how much of total data variability is mapped by each PC axis. So the first axis explains most overall variability, followed by number two, then three (if our data had three columns), and so forth. Note that the tail end of a highly-dimensional system will likely contain axes with an eigenvalue of zero (or close to zero). These map the system's **null space**, that is, they carry no additional information, and create a nasty singularity that will totally destroy subsequent results if given half a chance. Thus, if you're planning on using your principal components in follow-up procedures, it's an excellent idea to get rid of all bad apples with (near-)zero eigenvalues first.

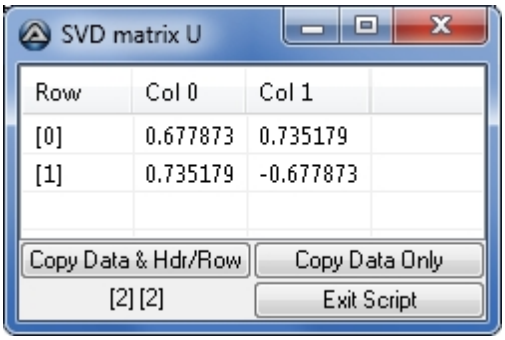
An easy way to evaluate the relative size of the eigenvalues is in terms of proportions of one (1), or as percentages:

```
$specID = 1    ; define the "sum" of all cell values

$total = _Eigen_MatrixSpecs_Single ( $matD, $specID )   ; store sum of all matrix cells in $total

_Eigen_CwiseScalarOp_InPlace ( $matD, "/", $total )      ; divide each cell by our sum
_MatrixDisplay ( $matD, "proportion var explained" )     ; show as proportions of one

_Eigen_CwiseScalarOp_InPlace ( $matD, "*", 100 )         ; or multiply each cell by 100,
_MatrixDisplay ( $matD, "% var explained" )              ; and show as percentages
```




These results tell us that the second PC dimension contributes only a small fraction (less than 4%) to total data variability. This will become important later on. For now, let's move on to the second major output of Jacobi SVD: the eigen**vectors** = left singular vectors = orthogonal unit vectors. Rows represent the relative contributions of each original variable to each principal component (variable's contribution = row, PC = column). The values per column are sometimes called the PC "coefficients" or "loadings". Their generic output container is matrix **U**.

```
$matU = _Eigen_GetActiveMatrix ( "U" )          ; Get the newly created matrix ID
_MatrixDisplay ( $matU, "SVD matrix U" )        ; show the eigenvectors
```

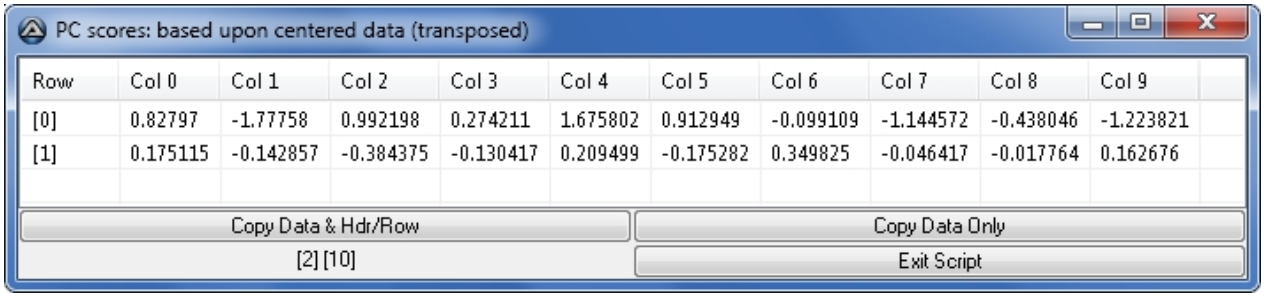



If you have large multivariate data sets, specific principal components *may* (if you are lucky) comprise major contributions from only one or a few original variables (that may themselves be related in some meaningful way). Since matrix **U**'s columns are (like the eigenvalues) ordered by size, this means that the most influential variables, i.e., the ones explaining most observed variability overall, are the ones with the largest coefficients in the left-most columns. Some scientific discoveries are made this way. In this case we're not so lucky, as all coefficient contributions are approximately equal per component.

Question: **why** do we need the eigenvectors (that is, matrix **U**)? Answer: left-multiplying SVD matrix **U** with the original data constitutes an orthogonal transformation into **size-ordered** principal component-space, in which the first axis represents the axis of largest variance, and consecutive axes map ever smaller residual data variability. The resulting coordinates are sometimes called the "PC scores" (don't worry if this still sounds cryptic).

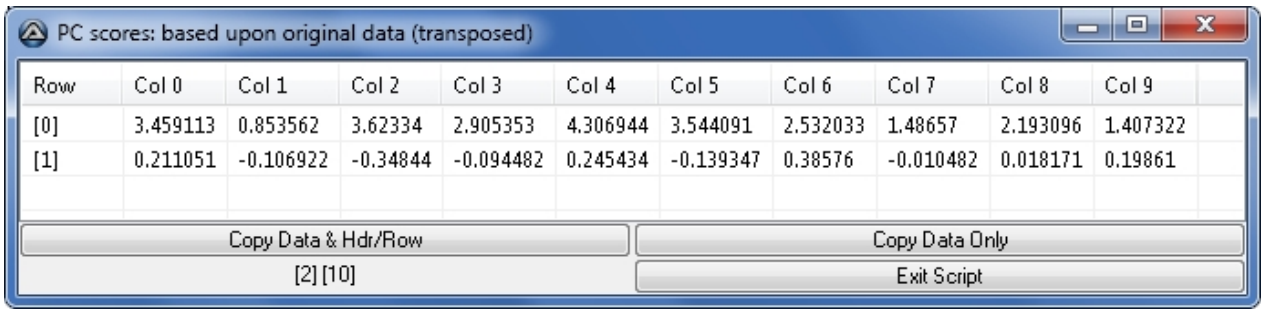
We can use the mean-centered data for this...

```
$matQ = _Eigen_Multiply_ABt ( $matU, $matB )  ; NB produces a transposed result!
_MatrixDisplay ( $matQ, "PC scores: based upon centered data (transposed)" )
```




...or we can use the original data:

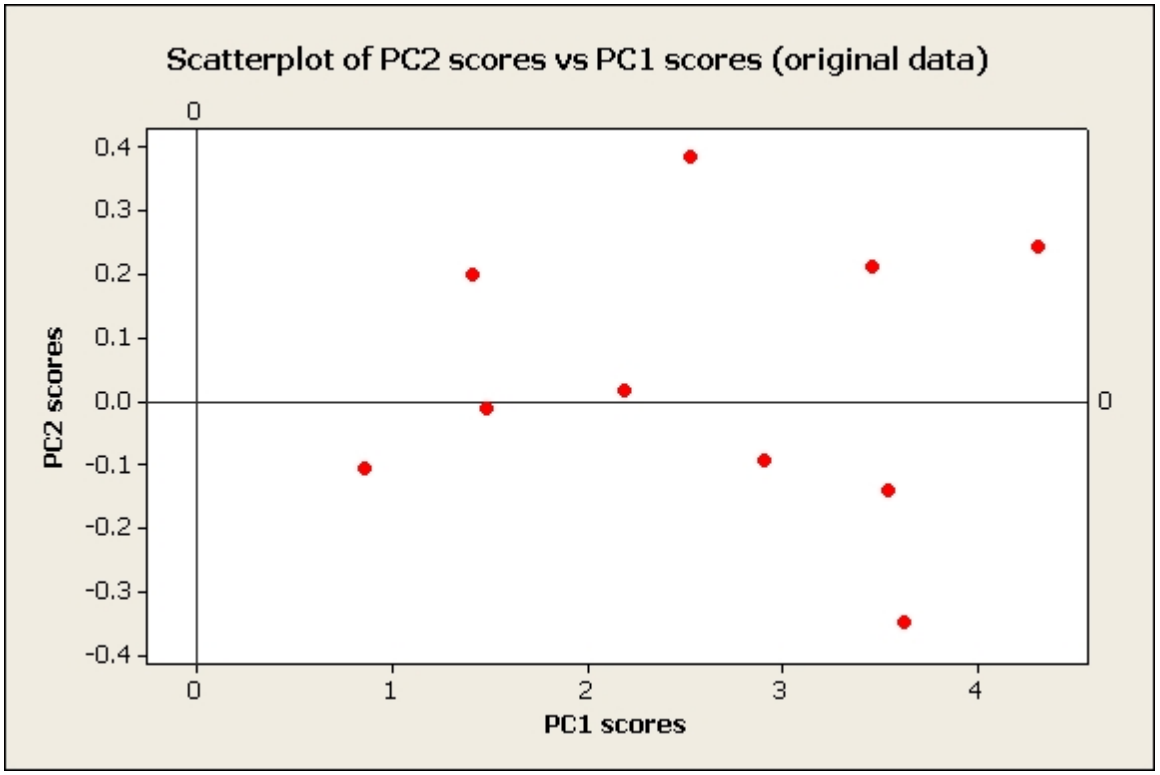
```
$matP = _Eigen_Multiply_ABt ( $matU, $matA )   ; produces a transposed result!
_MatrixDisplay ( $matP, "PC scores: based upon original data (transposed)" )
```




These two results produce the same spatial point cloud, except that matrix **P** = matrix **Q** translated (shifted) by [the column means multiplied by the eigenvectors], as is shown here (and check out the double-nested call in the snippet):

```
$matT = _Eigen_CloneMatrix ( $matQ )  ; create a test container

_Eigen_CwiseBinaryOp_Colwise_InPlace ( $matT, "+", _Eigen_Transpose ( _Eigen_Multiply_AB ( $matM,
$matU ) ) )      ; nested calls
_MatrixDisplay ( $matT, "matQ translated by M colwise" )

MsgBox ( 0, "IsCwiseEqual?", "Are these two results identical?" & @CR & _
        ( ( _Eigen_IsCwiseEqual_AB ( $matP, $matT, 1.0e-3 ) ) ? "Yeah, Baby!" : "No way, Bro!" ) )     ;
AutoIt's ternary operator
```

Note the difference in PC axis scales when we plot matrix **P**; the vertical range (PC2, range ca. 0.8) is now a small fraction of the horizontal range (PC1, range ca. 4), unlike in the original data plot, where they were about equal in size.




Congratulations! You have transformed your data into principal component space. Now what? What does this even mean?

Well, remember how the original input variables seemed mutually correlated? The new axes are all **orthogonal** (at right angles) to one another, and the points' coordinates along any particular axis are now **uncorrelated(!)** with their counterparts along **all** other axes. This means that we've effectively split the compound profile of the original, interdependent data into separate, **independent** contributing factors (= axes). Let that sink in for a moment.

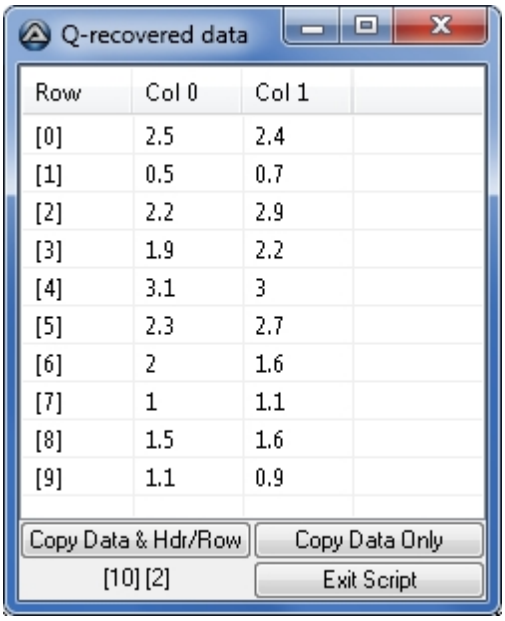
PCA is a **reversible** operation, meaning we can recover our original data from the PCs if we so desire. Because **U** is an orthogonal matrix, its inverse equals its transpose by definition: orthogonal $\mathbf{U}^{-1} = \mathbf{U}^{T}$, so $\mathbf{R} = \mathbf{P}^{T} * \mathbf{U}^{T}$. Instead of creating yet another matrix container just to store the transposed version of **U**, we'll call a dedicated Multiply function in the following code snippet that reads in the original matrices transposed directly, without first moving or copying the data inside them. To retrieve our original data, we multiply our PC-transformed points with the transpose of (eigenvector) matrix **U**, and, if using centered data, add our earlier-stored column means per column to each cell to de-center the coordinates again:

```
$matR = _Eigen_Multiply_AtBt ( $matP, $matU )  ; both matrices are multiplied in transposed form
_MatrixDisplay ( $matR, "P-recovered data" )

$matR = _Eigen_CwiseBinaryOp_Rowwise ( _Eigen_Multiply_AtBt ( $matQ, $matU ), "+", $matM )    ;
nested calls
_MatrixDisplay ( $matR, "Q-recovered data" )
```

Function _Eigen_CwiseBinaryOp_Rowwise() we've seen before when we subtracted the mean with the minus operator ("-"). This time, we're adding the mean back with the plus operator ("+"). It really is that simple! It's actually more complicated to do this in Eigen/C++ itself!

Incidentally, the nested call works like this:

1. _Eigen_Multiply_AtBt() returns the ID of a new, so far unnamed matrix;
2. _Eigen_CwiseBinaryOp_Rowwise() alters that matrix with the contents of matrix **M** and assigns the former's ID (with altered contents) to variable $matR (so we can handle matrix **R** thereafter).




Still with me? Because now comes THE IMPORTANT PART! (fanfare, dancing elephants, fireworks...)

Recall that we earlier established that the second principal component (PC2) contributed very little to overall data variability. That means that if we discard it, we won't lose much information. This is often called "dimensional reduction." It is used in data compression and facial recognition, for example. Common practice is to discard all PC's with eigenvalues below unity (the value 1), or to plot eigenvalue versus principal component rank and look for a steep drop-off. We can also take the cumulative percentage of total variability explained by adding all cell values in matrix **D** (a vector) up to cell (PC) 2, 3, 4, and so on, until we reach our target of X% of total variance mapped. The target may be >50%, or 90%, or 95%, for example. The choice of cut-off depends on the data context and your own aims and judgement. Seek thee Wisdom!

We can discard principal components by zeroing out their column in matrix **U**. This is even easier than it sounds. In our simplified example, we'll discard PC2, keeping only PC1. Generally speaking, please be careful when destroying data like this; make certain you won't be needing the discarded columns of matrix **U** in future, or else make a backup copy of **U** beforehand (nag nag nag).

```
_Eigen_SetZero_Col ( $matU, 1 )          ; colindex = base-0, so PC2 = column 1!
```

Finally, we retrieve the **new** dataset, expressed in the **original** reference frame! Here we use nested Eigen calls again:

```
$matR = _Eigen_CwiseBinaryOp_Rowwise ( _Eigen_Multiply_AtBt ( $matQ, $matU ), "+", $matM )
_MatrixDisplay ( $matR, "dimension-reduced" )
```

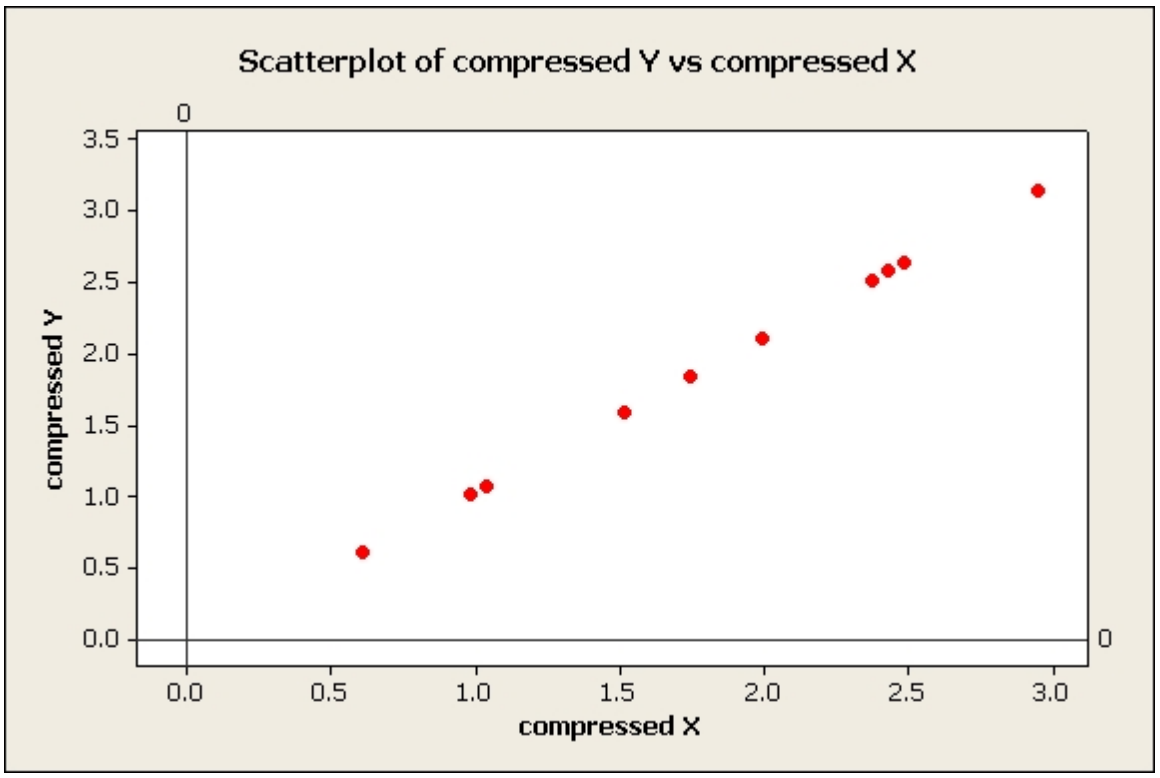
dimension-reduced

| Row | Col 0 | Col 1 |
|---|---|---|
| [0] | 2.371259 | 2.518706 |
| [1] | 0.605025 | 0.603161 |
| [2] | 2.482584 | 2.639443 |
| [3] | 1.99588 | 2.111594 |
| [4] | 2.945981 | 3.142014 |
| [5] | 2.428864 | 2.581181 |
| [6] | 1.742816 | 1.837137 |
| [7] | 1.034125 | 1.068535 |
| [8] | 1.51306 | 1.587958 |
| [9] | 0.980404 | 1.010273 |

Copy Data & Hdr/Row | Copy Data Only

[10] [2] | Exit Script

The data reconstruction derived from the single remaining eigenvector is now neatly aligned, while retaining over 96% of the original variability. The resulting graph of our new dataset shows that the back-projected point cloud has lost the secondary scatter at right angles to its main axis, while retaining the full spread along its slope. This is called "lossy compression," used, for example, to turn .wav sound files (as stored on music CDs) into .mp3 files (which, on average, require about one tenth the space of a wav file; now you know why).




Obviously, the full power of PCA only becomes apparent once you are dealing with large data sets with dozens, thousands, perhaps even millions of variables (= dimensions). For example, in facial recognition, each same-sized B/W image can be stored as a single row vector of pixel grayscale values (in which the original pixel rows of the image are simply appended in sequence). Suppose we then compare 50 images of different faces, and each image is 1000 x 1500 pixels. Then our input matrix has 50 rows and 1.5 million

columns (variables) = 75 million cells (note: special ma-tricks exist to reduce this huge multi-dimensional space prior to computation).

PCA may then establish "the common face" defined by, say, the first few hundred principal components, and then express the original images in terms of those PCs only (lossy compression), or attempt to identify a new image by finding the nearest match in terms of the PC profile of the faces in the database (facial recognition). Google the term "Eigenface" to find out more. Issues concerning differences in scale, image quality, lighting, angle, and facial expression complicate such efforts no end, so I would suggest you start exploring PCA with a more modest problem.

Now that you've worked through the steps and examined the various outputs, PCA hopefully no longer seems a blackbox algorithm anymore. You are now fully prepared to start using the two in-built functions _Eigen_PCA() and _Eigen_PCA_ReDim() directly. The latter has parameter $PCs_to_Keep as input and spits out matrix **R**; If $PCs_to_Keep equals the number of original variables or is left at zero (the default), you will retrieve your original data set (useful as a test only). Enabling **P** and/or **Q** suffices here; the rest is activated as needed (flags-dependent). In the following snippet (the start of a new script), we omit duplicating the first few lines of the Tutorial script, where we initialise the work environment and load matrix **A** with the original data (see first snippet).

```
_Eigen_SetActiveMatrix ( "PQ", True )     ; True = reset all flags first

_Eigen_PCA ( $matA )    ; rather shorter than before, no?
If @error Then Exit      ; Can I go now?
```

Alternatively, you could have used the provided flags $returnPscores and $returnQscores, like this:

```
_Eigen_ResetActiveMatrix()       ; we're not using these switches now; this function disables them all

$center = True                   ; set some operational booleans
$normalise = False
$computeCovar = True
$returnPscores = True            ; set this True if you want matrix P returned
$returnQscores = True            ; set this True if you want matrix Q returned

_Eigen_PCA ( $matA, $center, $normalise, $computeCovar, $returnPscores, $returnQscores )
If @error Then Exit

$matB = _Eigen_GetActiveMatrix ( "B" )  ; You can store these IDs anywhere you like
$matC = _Eigen_GetActiveMatrix ( "C" )  ; (they are just indices to array $MatrixList),
$matD = _Eigen_GetActiveMatrix ( "D" ) ; but here we'll use sensible variable names.
$matM = _Eigen_GetActiveMatrix ( "M" );            ...
$matP = _Eigen_GetActiveMatrix ( "P" )  ; "I like New York in June...."
$matQ = _Eigen_GetActiveMatrix ( "Q" ) ; How about you?" (ta dum dum dum),
$matS = _Eigen_GetActiveMatrix ( "S" )  ; "I like a Gershwin tune...."
$matU = _Eigen_GetActiveMatrix ( "U" ) ; How about you?" (ta dum dum)

_MatrixDisplay ( $matB, "centered data" )          ; display the various results excreted by _Eigen_PCA()
_MatrixDisplay ( $matC, "covariance matrix" )
_MatrixDisplay ( $matD, "PC# proportional contributions" )
_MatrixDisplay ( $matM, "column means" )
_MatrixDisplay ( $matP, "transformed original data (transposed)" )
_MatrixDisplay ( $matQ, "transformed centered data (transposed)" )
```

```
_MatrixDisplay ( $matS, "eigenvalues" )
_MatrixDisplay ( $matU, "eigenvectors" )
```

After calling _Eigen_PCA() once, we can try different PC transforms as often as needed, without calling _Eigen_PCA() over and over again. We can use either **P**, or **Q** and **M** (both!) as inputs, but not all at the same time. Here is the matrix **P**-based data recovery test:

```
$PCs_toKeep = 0         ; use all PCs (recovery testing)

$matR = _Eigen_PCA_ReDim ( $matU, $PCs_toKeep, 0, $matP )     ; 0 = $matM is not used here
If @error Then Exit

Local $arrayR               ; let's capture the results *directly* in an array this time
_Eigen_CreateArray_FromMatrix ( _Eigen_GetActiveMatrix ( "R" ), $arrayR )
_ArrayDisplay ( $arrayR, "recovered via P" )
```

And here's the **QM**-based recovery test (only the second and last line are different):

```
$PCs_toKeep = 0         ; use all PCs (recovery testing)

$matR = _Eigen_PCA_ReDim ( $matU, $PCs_toKeep, $matM, 0, $matQ )    ; 0 = $matP is not used here
If @error Then Exit

Local $arrayR  ; a bit redundant perhaps (see above)
_Eigen_CreateArray_FromMatrix ( _Eigen_GetActiveMatrix ( "R" ), $arrayR )
_ArrayDisplay ( $arrayR, "recovered via QM" )
```

Actual **dimensional reduction** is achieved by setting $PCs_toKeep to some integer value larger than zero but smaller than the number of columns of the input matrix. In our simplified example, we started out with two axes, so we can only discard PC2 here, keeping one axis.

```
$PCs_toKeep = 1         ; At last, dimensional reduction!

$matR = _Eigen_PCA_ReDim ( $matU, $PCs_toKeep, $matM, 0, $matQ )    ; 0 = $matP is not used here
If @error Then Exit

_MatrixDisplay ( _Eigen_GetActiveMatrix ( "R" ), "lossy compression" )      ; matrix ID not stored
```

Finally, we release all matrices created during this session, and close the dll handle. Check out the reference links at the top of this Tutorial if you wish to learn more. Good night, and good luck!

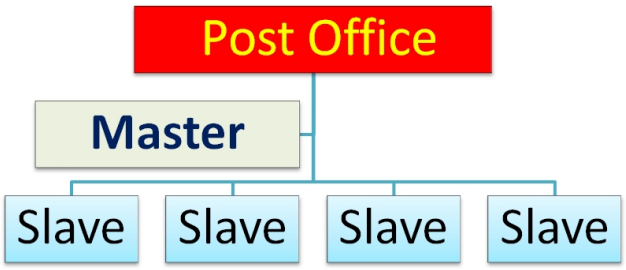
```
_Eigen_Cleanup ()

MsgBox ( 0, "Eigen4AutoIt", "This concludes this Tutorial." )
```

## Pooled Multi-Processing

# Tutorial - Pooled Multi-Processing

This tutorial explains the basics of multi-processing with **Eigen4AutoIt** (**E4A**), using **AutoIt**'s **Pool** environment (also developed by yours truly). **Pool** creates an infrastructure for multiple processes to communicate with each other, by means of messages, profile descriptions, executable instructions, and data transfers. These processes can be running on many different machines across a Local Area Network (LAN or WLAN), or on a single localhost (a single, stand-alone machine)., which is the simplest possible case, and the one we'll be exploring here.

The basic set-up of the **Pool** environment for our current purposes looks like this:




Unlike all previous tutorials, we'll be managing several processes at the same time, and these will share the load of a large computation or our astoundingly dumb toy example. **Pool** is an acronym for "**P**ost **O**ffice for **O**rganised **L**abour." Each physical machine that is part of a **Pool** runs a single instance of a ***Post Office*** script or executable. All other **Pool** processes on that machine hook up with that Post Office, and use it as a hub to send data and commands to other **Pool** processes, and to be kept informed on **Pool** membership changes. On a LAN or WLAN, Post Offices on different machines also exchange data with each other, and relay information between distant processes.

The ***Master*** node is the most powerful one in the depicted scenario; it launches its own Post Office if none is already running, and can then command it to start and manage several ***Slave*** processes (four in this case). Unlike the Post Office, both Master and Slaves can perform **E4A** operations, but Slaves are unaware of other Slaves in the **Pool**; they know of the existence of Master and Post Office only, and do not communicate with anyone else. Once they are up and running, they simply wait for data transfers (via the Post Office) and a job description (directly from the Master). Once the job is done, the Master will command each Slave to return their output(s), and to leave the **Pool** thereafter. The Master then pieces the final result together, and the shared job is done.

A typical **E4A** Slave process comprises only a few dozen lines of standard code. It starts, like all **Pool** entities, by defining the type of **Pool** member it is going to be:

```
#NoTrayIcon
#include "C:\AutoIt\Eigen\Eigen4AutoIt.au3"      ; IMPORTANT: edit this path to match your Slave's E4A environment, and recompile!
#include ".\Pool.au3"    ; somewhat essential
```

```
; set attributes (fixed)
$_POOL_ATTRIB_CLASS = "E4A"           ; cannot be empty
$_POOL_ATTRIB_OPERATOR = "CLN"        ; CLN = Pool Client
$_POOL_ATTRIB_SLAVE = True            ; Your wish is my command, oh Master!
$_POOL_ATTRIB_SHAREDATA = True        ; T: handle incoming/outgoing vars, arrays, structs
$_POOL_ATTRIB_COLLECTOR = False       ; no need here
$_POOL_ATTRIB_LAUNCH_MGR = False      ; no need here
$_POOL_ATTRIB_CLOSEDOWN_UDF = "_Eigen_CleanUp"  ; extra tasks to perform when closing down

; set/disable relevant timeouts
$_POOL_ATTRIB_TIMEOUT_CLN_SOLO = 10 * 1000        ;  positive value = msec without MGR before
closing down
$_POOL_ATTRIB_TIMEOUT_INACTIVE = 10 * 60 * 1000   ; positive value = msec without msg/mail/
TODOlist before closedown
$_POOL_STATUS_TIMEOUT_UNCONDITIONAL = -1          ; positive value = msec before leaving Pool,
regardless of state
$_POOL_ATTRIB_TIMEOUT_CHORES = 750                ; nothing else to do, so reduce the default idle
interval

; set some status vars (dynamic)
$_POOL_STATUS_CHAT_ENABLED = False        ; No talking on the job!
$_POOL_STATUS_EXEQ_ENABLED = True         ; crucial!
$_POOL_STATUS_EXEQ_ENCRYPTED = False      ; for clarity and speed
$_POOL_STATUS_STAYALIVE = False           ; F: terminate when Pool is gone

; Jump in, the water's fine...
_Pool_StartUp ()
```

Even if you are totally unfamiliar with **Pool**, it should be clear what we're doing here: setting some fixed attributes, adjusting some timeouts, and switching various capabilities on or off. Please consult the **Pool** documentation for details. The only crucial flags for a Slave are $_POOL_ATTRIB_SLAVE, $_POOL_ATTRIB_SHAREDATA, and $_POOL_STATUS_EXEQ_ENABLED, which all have to be True for a Slave to be able to handle data transfers and carry out instructions sent to it by others.

The second step is for the Slave to set up an **Eigen4AutoIt** environment, which is a little different from what we did in previous tutorials:

```
$EIGEN_DLLPATH = "C:\AutoIt\Eigen"    ; please edit this path to reflect the target machine's Eigen
environment (and recompile!)

_Eigen_Hide_Errors ()          ; these would appear in a (msg-blocking!) MsgBox and wait indefinitely
for user-input
_Eigen_Hide_Warnings ()        ; ditto
_Eigen_DebugMode_Off ()        ; Fire up the Eigen engine (you can restart this Slave with different E4A
settings via the ExeQ later)
```

Firstly, as of **Eigen4AutoIt** version **2.4**, users can change $EIGEN_DLLPATH between the **E4A** #include statement and calling **_Eigen_Startup()**, so as to reflect the location of the **E4A** environment on a particular machine without having to edit ***Eigen4AutoIt.au3*** everywhere. Be sure to recompile ***E4Aslave.au3*** if you edit this path, because the Post Office will run the Slave executable, not the Slave's source script. Secondly, all **E4A** errors and warnings are to be switched off because they would suspend execution, which would cause two potential problems. If the Slave is running on a different machine, there likely won't be a user sitting in front of its screen to press <Ok> (the machine may not even have a screen or keyboard!), and secondly, using a MsgBox or any other blocking function (InputBox, Beep...) is a really bad idea in an event-driven application such as a **Pool** Slave, because you will probably start losing important system and **Pool**

messages while Windows message-handling is suspended during the blocking. Instead, **Pool** Slaves will tell their Master directly when something goes wrong, if they can.

Finally, the Slave starts its **Eigen** engine, not with **_Eigen_Startup()** but with **_Eigen_DebugMode_Off()**, which itself calls **_Eigen_Startup()** after switching to Computing Mode. This means we won't be getting any assert failure messages from **Eigen**'s dlls either, and matrix operations will be carried out at top speed, without extensive error and bounds checking. So it is probably worth testing the operations you wish your Slaves to perform on a small sample dataset beforehand, to ensure no such errors will be generated.

Now comes the hard part, where the poor, fettered Slaves are cruelly exploited by the evil Master, having to working so hard and for so long that they can hardly... wake up?

```
While True
        Sleep ( 10000 )
WEnd

_Pool_CloseDown ()
```

Slave processes are purely message- and event-driven, so they don't do anything unless you tell them to. The rest of the time they conserve their strength. And that's al there is to a Slave. Moving right along.

The **Post Office** script (not shown) looks similar to that of a Slave, with a few exceptions. For one, the **Eigen4AutoIt** environment is absent, because the Post Office does not do computations; it is already plenty busy relaying data and messages between other processes. Various **Pool** settings are also different: its operator type is Pool Manager (MGR, instead of CLN for Pool clients), it is working without a **Pool** Server for sending data packets around a network ($_POOL_STATUS_MGR_SOLO = True), it is allowed to launch new client processes itself ($_POOL_ATTRIB_LAUNCH_CLN = True), and these processes are called **E4ASlave** ($_POOL_ATTRIB_CLN_DEFAULT = "E4ASlave.exe"). After defining these Pool settings, a Post Office remains indefinitely in a wait-and-sleep loop. Like the Slaves, it too is purely event- and message-driven.

We are working our way up the chain of command: the Post Office will launch the Slaves, and before that, the Master launches the Post Office (unless one was already running on you machine), through settings $_POOL_ATTRIB_LAUNCH_MGR = True and $_POOL_ATTRIB_MGR_DEFAULT = "PostOffice_E4A_Solo.exe". One other attribute worth mentioning is that all processes (Master, Slaves, and Post Office) have to be of the same Pool Class: $_POOL_ATTRIB_CLASS = "E4A". It is possible to run several pools concurrently on the same machine(s), and their class is what distinguishes one set of communicating processes from another. We'll skip the other **Pool** settings of the Master script (see the source code for this), and jump right in where the real action starts:

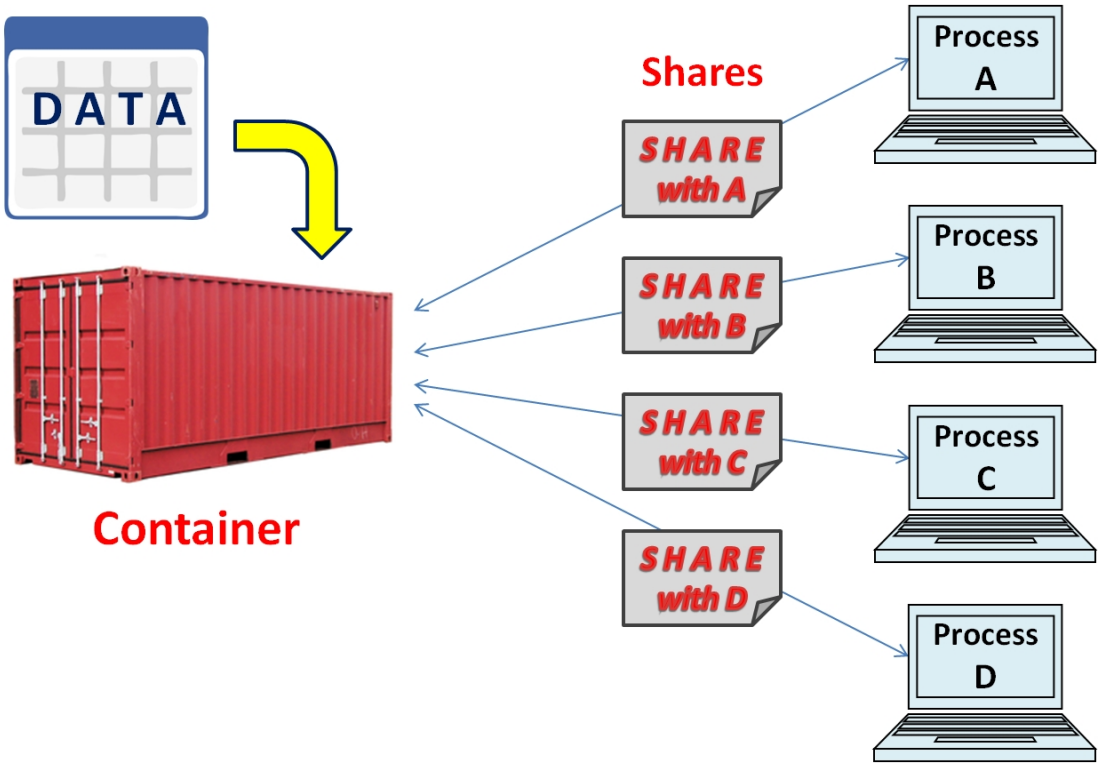
```
_Pool_Startup ()
$_POOL_STATUS_HOOKEDUP_UDF = "_HookUp_Tasks"
```

**Pool** has a number of status variables with the suffix "_UDF". You can fill these with the name of your own function you wish **Pool** to execute whenever certain circumstances occur. Because $_POOL_ATTRIB_LAUNCH_MGR = True, the **Pool** startup procedure will check whether a Post Office of the Master's class is already running, and if not, launch the excutable defined in $_POOL_ATTRIB_MGR_DEFAULT. Subsequently, the Master script willl repeatedly attempt to hook up with this local Post Office, which involves the exchange of **Pool** member profiles and various initial status messages so each side knows whom they are dealing with. Once completed, the hook-up procedure will call any valid, accessible UDF defined in $_POOL_STATUS_HOOKEDUP_UDF, which in this case, leads to the function definition of **_HookUp_Tasks()** further down:

```
Func _HookUp_Tasks ()

        ; Start four E4A slave processes (regardless of how many CPU cores are available)
        _Pool_Send_Command ( $_POOL_STATUS_MGR_ID, "AddMembers", 4 )

        ; Tell the Post Office to reserve eight data containers for us;
        _Pool_Send_Command ( $_POOL_STATUS_MGR_ID, "RequestContainerIDs", 8 )
EndFunc
```

Here we see the most important way scripts control **Pool** actions, by calling **_Pool_Send_Command().** This function always expects a **Pool** target ID and a Pool command (a string without spaces), and in these two cases, also a single parameter. Normally, target IDs can be looked up in the $_Pool_MemberList array (PML), but the local Post Office (= Pool manager) is also stored in status variable $_POOL_STATUS_MGR_ID. Note that target IDs are machine-specific; each Post Office maintains its own list, assigning its own IDs in the order new member profiles are received locally. So on a LAN, process A may have one ID on its own machine, but a different number on any other machine. All messages between different machines therefore involve a conversion of IDs to their local equivalent, to avoid confusion.

Function **_HookUp_Tasks()** does two things. The first Pool command tells the Post Office to start running four Slave executables, whereas the second one requests eight new Container IDs for future use (temporarily stored in internal **Pool** array $_Pool_ReservedContainerIDs). Shipping Containers are the vessels **Pool** uses to transfer chunks of data. It doesn't matter whether this concerns an array, a list of variables, a file, or, in our case, an **E4A** matrix struct; they all get the same type of Container, assigned with its own ID number by the local Post Office. In order to get the Container shipped anywhere, a Pool entity additionally has to define at least one **Share**. A single Container can have many Shares for as many different recipient processes (see figure). A Share is like a shipping manifest; it defines where a copy of the Container is to be sent, and/or from whom it is to be retrieved. Subsequently, whenever a Container's creator receives Pool command "LoadSendContainer," it will update the Container's local contents with the current version of the associated local data, and then initiate a data transfer for each defined Share. Thus a single outward transfer is like a broadcast for all defined destinations. Contrastingly, if the same Pool command "LoadSendContainer" is sent to an individual recipient (or "Share target"), it alone will load *its* local Container with *its* local data and initiate a *single reverse* transfer, back to the Container's creator.

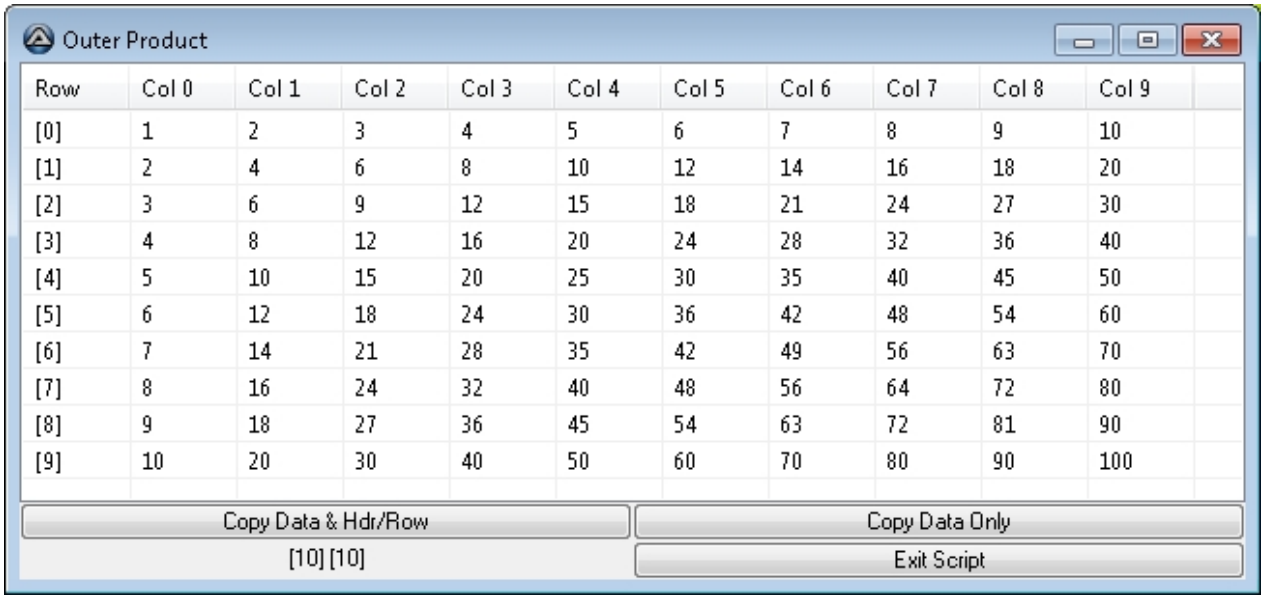



One other peculiar feature of Pool Containers, one that is especially relevant in the context of **Eigen4AutoIt**, is that once a Sharing relationship is defined between creator and recipient, the associated Container can be returned *before it is sent out*. In fact, it need never be sent out at all. Creating a Share causes the recipient to allocate memory for the specified data size and to associate this *struct* with a local variable. Now take a moment to check out function _Eigen_Add_ExistingMatrix() and you'll realise that if we can tell a Slave to execute this function in its own environment, we can remotely map any matrix at that destination to a Container we defined earlier, let the Slave store a computation result in it, and then tell it to return it to us pronto. Now it finally makes sense why most **E4A** functions allow optional matrix IDs to be parsed to define existing output matrices! It's not just to conserve memory resources of the main programme, but foremost to enable this remote processing trick to work. Finally, the secret's out.

Enough theory; once more into the fray. The Master programme will start by performing a simple matrix operation all by itself, and then achieve the same result by breaking up the problem into four equal parts, first by itself (to illustrate the operation-in-parts), and then farmed out in those four parts to the four Slaves, to demonstrate actual multi-processing. The selected operation is _Eigen_OuterProduct(), which takes as inputs a col vector and a row vector (in any order), and produces a multiplication table of these two (a matrix with as many rows as the col vector and as many cols as the row vector):

```
_Eigen_DebugMode_Off ()         ; calls _Eigen_Startup()
$_POOL_ATTRIB_CLOSEDOWN_UDF = "_Eigen_CleanUp"

$size = 10       ; this should be an even number

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 1, $size, 1, $size )   ; row vector
_MatrixDisplay ( $matA, "Rowvector A" )

$matB = _Eigen_CreateMatrix_LinSpaced_ColMajor ( $size, 1, 1, $size )    ; col vector
_MatrixDisplay ( $matB, "Colvector B" )

$matR = _Eigen_OuterProduct ( $matA, $matB )
_MatrixDisplay ( $matR, "Outer Product" )
```

As both vectors contain the integers from 1 to 10, the result is the 10x10 multiplication table we were all tortured with when we were little:




| Row | Col 0 | Col 1 | Col 2 | Col 3 | Col 4 | Col 5 | Col 6 | Col 7 | Col 8 | Col 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| [0] | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| [1] | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| [2] | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| [3] | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| [4] | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| [5] | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| [6] | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| [7] | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| [8] | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| [9] | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

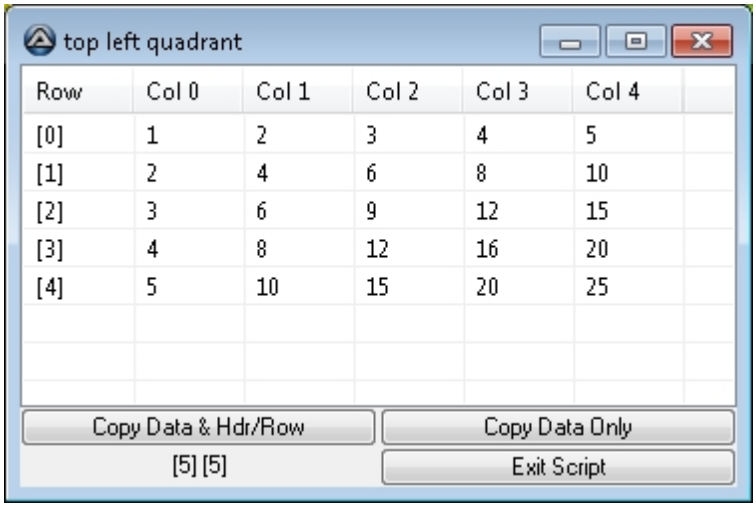
Now imagine we take a pair of scissors and cut this childhood trauma into two equal halves horizontally, and again vertically. Each resulting quadrant can be considered a 5x5 matrix of its own, and could be generated by calling **_Eigen_OuterProduct()** with the associated halves of the two original input vectors, which we can produce like this:

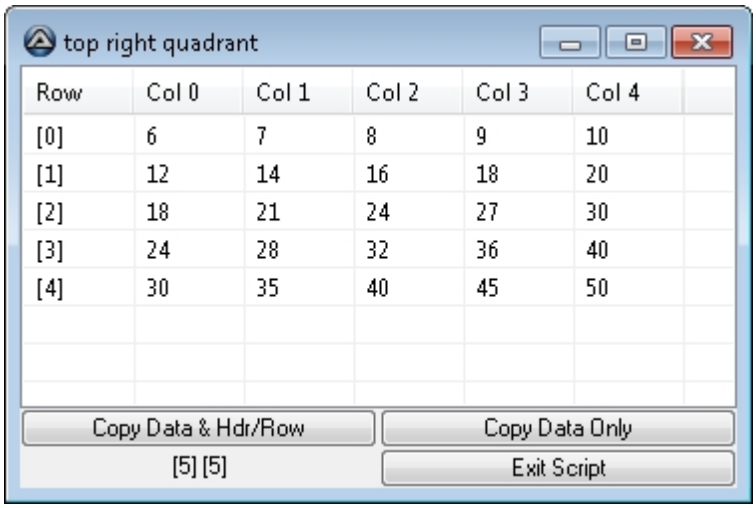
```
$halfsize = $size * 0.5

_Eigen_SplitMatrix_FromAcol ( $matA, $halfsize )
$matA_left = _Eigen_GetActiveMatrix ( "B" )
$matA_right = _Eigen_GetActiveMatrix ( "C" )

_Eigen_SplitMatrix_FromArow ( $matB, $halfsize )
$matB_top = _Eigen_GetActiveMatrix ( "B" )
$matB_bot = _Eigen_GetActiveMatrix ( "C" )
```

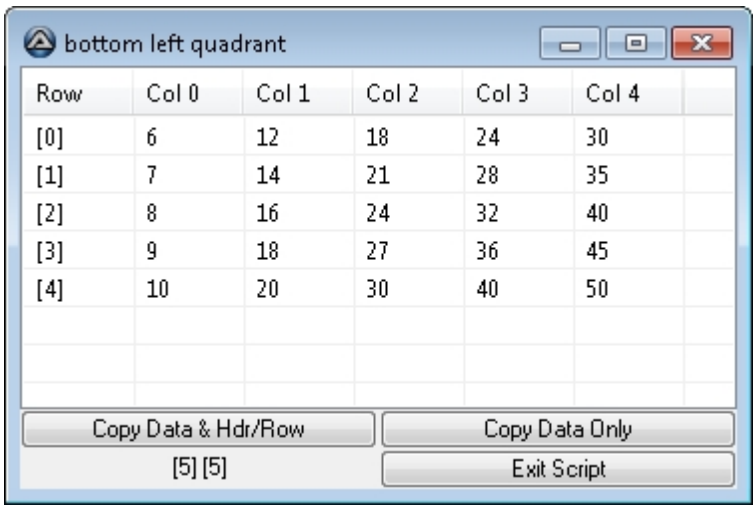
And once we've got our four hobbit-sized input vectors, we can create four matching hobbit-sized quadrant matrices with four **_Eigen_OuterProduct()** calls:

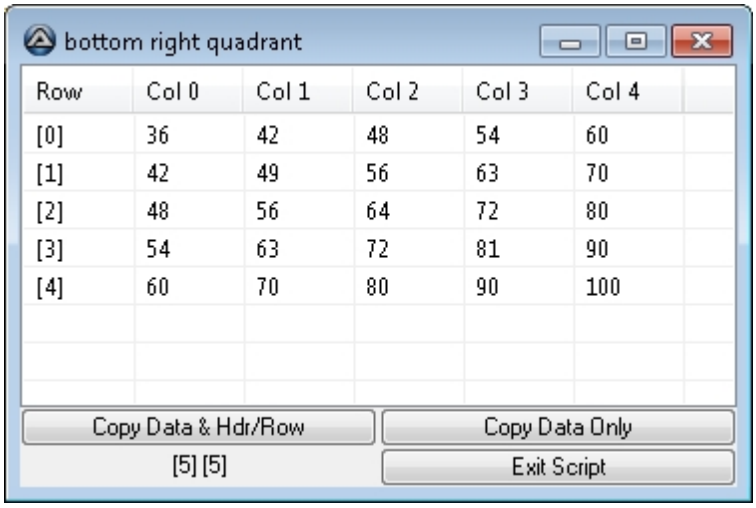
```
$matR_topleft = _Eigen_OuterProduct ( $matA_left, $matB_top )
_MatrixDisplay ( $matR_topleft, "top left quadrant" )

$matR_topright = _Eigen_OuterProduct ( $matA_right, $matB_top )
_MatrixDisplay ( $matR_topright, "top right quadrant" )

$matR_botleft = _Eigen_OuterProduct ( $matA_left, $matB_bot )
_MatrixDisplay ( $matR_botleft, "bottom left quadrant" )

$matR_botright = _Eigen_OuterProduct ( $matA_right, $matB_bot )
_MatrixDisplay ( $matR_botright, "bottom right quadrant" )
```

Try to hide your intense surprise and delight as we proudly present our four mini-matrices:


top left quadrant
Row Col 0 Col 1 Col 2 Col 3 Col 4
[0] 1 2 3 4 5
[1] 2 4 6 8 10
[2] 3 6 9 12 15
[3] 4 8 12 16 20
[4] 5 10 15 20 25
Copy Data & Hdr/Row
Copy Data Only
[5] [5]
Exit Script


top right quadrant
Row Col 0 Col 1 Col 2 Col 3 Col 4
[0] 6 7 8 9 10
[1] 12 14 16 18 20
[2] 18 21 24 27 30
[3] 24 28 32 36 40
[4] 30 35 40 45 50
Copy Data & Hdr/Row
Copy Data Only
[5] [5]
Exit Script


bottom left quadrant
Row Col 0 Col 1 Col 2 Col 3 Col 4
[0] 6 12 18 24 30
[1] 7 14 21 28 35
[2] 8 16 24 32 40
[3] 9 18 27 36 45
[4] 10 20 30 40 50
Copy Data & Hdr/Row
Copy Data Only
[5] [5]
Exit Script




Finally, what was rent asunder shall be reunited again by the near-limitless power vested in **_Eigen_Copy_Ablock_ToBblock()**:

```
_Eigen_SetZero ( $matR )          ; just to show we're not cheating

_Eigen_Copy_Ablock_ToBblock ( $matR_topleft, $matR, 0, 0, $halfsize, $halfsize, 0, 0 )
_Eigen_Copy_Ablock_ToBblock ( $matR_topright, $matR, 0, 0, $halfsize, $halfsize, 0, $halfsize )
_Eigen_Copy_Ablock_ToBblock ( $matR_botleft, $matR, 0, 0, $halfsize, $halfsize, $halfsize, 0 )
_Eigen_Copy_Ablock_ToBblock ( $matR_botright, $matR, 0, 0, $halfsize, $halfsize, $halfsize, $halfsize )
_MatrixDisplay ( $matR, "Outer Product, 4 quads = same result" )          ; tadaaah!

$matRlocal = _Eigen_CloneMatrix ( $matR )       ; for future reference
```

In the final line of this snippet we create a copy to compare against the result of the multi-processing, to which we are finally turning now.

While we were frolicking with matrices, the aforementioned settings $_POOL_ATTRIB_LAUNCH_MGR = True and $_POOL_ATTRIB_MGR_DEFAULT = "PostOffice_E4A_Solo.exe" will have launched a local Post Office, which by now should have hooked up with our Master script, which implies that the Master's Pool variable $_POOL_STATUS_MGR_ID is now filled with ID 1 (the local Post Office is by definition always assigned ID 1, and $_POOL_STATUS_MGR_ID = -1 (minus one) if it is absent). This provides us with an easy way of checking whether **Pool** facilities are currently available. Before we get going, we also need to confirm that we have by now received our eight requested Container IDs. and if not, wait a bit until we do (snippet skipped, see script for details).

We will start by creating four brand-new, empty results matrices called quadrant1 to quadrant4, and then provide Containers for these four, and for the four half-sized vectors that we used to generate the four quadrant results. We use **Pool** function **_Pool_Container_Create()** for this, which expects the variable itself (the struct in column zero of the $MatrixList), the type of variable (in a string, in this case "struct"), an optional variable name (left blank for **E4A** matrices), and its dimensions (rows and cols). If all goes well, the function returns an index to **Pool**'s local list of defined Container structs $_Pool_StructList, just like a matrix ID returned by **_Eigen_CreateMatrix()** is an index to an entry in $MatrixList.

```
; create partial output buffers of the right dimensions
$quadrant1 = _Eigen_CreateMatrix_Zero ( $halfsize, $halfsize )
$quadrant2 = _Eigen_CloneMatrix ( $quadrant1 )
$quadrant3 = _Eigen_CloneMatrix ( $quadrant1 )
$quadrant4 = _Eigen_CloneMatrix ( $quadrant1 )

; create eight Containers
```

```
$index_q1 = _Pool_Container_Create ( $matrixList[$quadrant1][0], "struct", "", _Eigen_GetMatrixRows
( $quadrant1 ), _Eigen_GetMatrixCols ( $quadrant1 ) )
$index_q2 = _Pool_Container_Create ( $matrixList[$quadrant2][0], "struct", "", _Eigen_GetMatrixRows
( $quadrant2 ), _Eigen_GetMatrixCols ( $quadrant2 ) )
$index_q3 = _Pool_Container_Create ( $matrixList[$quadrant3][0], "struct", "", _Eigen_GetMatrixRows
( $quadrant3 ), _Eigen_GetMatrixCols ( $quadrant3 ) )
$index_q4 = _Pool_Container_Create ( $matrixList[$quadrant4][0], "struct", "", _Eigen_GetMatrixRows
( $quadrant4), _Eigen_GetMatrixCols ( $quadrant4 ) )

$index_left = _Pool_Container_Create ( $matrixList[$matA_left][0], "struct", "", _Eigen_GetMatrixRows
( $matA_left ), _Eigen_GetMatrixCols ( $matA_left ) )
$index_right = _Pool_Container_Create ( $matrixList[$matA_right][0], "struct", "", _Eigen_GetMatrixRows
( $matA_right ), _Eigen_GetMatrixCols ( $matA_right ) )
$index_top = _Pool_Container_Create ( $matrixList[$matB_top][0], "struct", "", _Eigen_GetMatrixRows
( $matB_top ), _Eigen_GetMatrixCols ( $matB_top ) )
$index_bot = _Pool_Container_Create ( $matrixList [$matB_bot][0], "struct", "", _Eigen_GetMatrixRows
( $matB_bot ), _Eigen_GetMatrixCols ( $matB_bot ) )
```

**_Pool_Container_Create()** does not take a matrix ID as argument (which would have been a lot simpler here) because its application is not restricted to **E4A** environments, and several other variable types need to be accommodated as well (see again the **Pool** documentation).

The next step is to create **Share** definitions, to associate each Container with one or two destination processes. But for this to work, our Slaves do need to be up and running, registered with the Post Office, and listed in the Master's own $_Pool_MemberList. As a Pool initialisation can take several seconds, we will hang around in a waiting loop while repeatedly listing all other Pool client IDs (CLNs) currently available, with **_Pool_ListAllOtherCLN_IDs()**., which returns a 1D-array of local client Pool IDs other than your own ("local" meaning they run on the same machine as your current script). As soon as four CLNs have registered (which we know to be our Slaves), we move on. If something went horribly wrong, and at least one Slave never came online, we abort instead by calling **_Pool_Closedown()**. You should never terminate a **Pool** application by calling Exit directly, as various open handles need to be closed first, else you're likely to experience a very hard landing.

```
$onLocalhost = True    ; F: on entire LAN, T: on this machine (localhost)
$tstart = TimerInit ()
While True
        Global $IDlist = _Pool_ListAllOtherCLN_IDs ( $onLocalhost )          ; excludes self
        If $IDlist[0] = 4 Then ExitLoop

        If TimerDiff ( $tstart ) > 30000 Then
                MsgBox ( 16, "Eigen4AutoIt HTC demo", "By now, the Post Office should " & _
                "have launched four slaves, and assigned all Container IDs,  so something went wrong.
Aborting..." )
                _Pool_Closedown ()
        EndIf
        Sleep ( 1000 )
WEnd
```

If we get beyond this point, our four Slaves have awoken and await instructions. We start by defining their Shares, because that will take most time, and we can continue preparing the rest of the operations while Share processing at the destination is still ongoing. We create Shares with the unimaginatively named function **_Pool_Container_CreateShare()**, which expects a target ID (the ones we've already collected in our $IDlist array) and a Container ID.

Now when we called **_Pool_Container_Create()**, the earlier-requested Container IDs were automatically

assigned to each of the eight matrices we thus packaged. However, at this point, we don't know *which* ID our Post Office has assigned to which Container. What we do have, however, is each Container's index, and it's trivial to retrieve its ID from that, using **_Pool_PSLindexToStructID()**., "PSL" being short for $_Pool_StructList, the array in which all local Pool Containers are stored, and structID = Container ID. In the following snippet, we never actually store Container IDs, but parse them on directly to **_Pool_Container_CreateShare()** using the Container indices.

```
; create four shares for the results matrices first, so Containers 1-4 match the results to be produced by Slaves 1-4 resp.
_Pool_Container_CreateShare ( $IDlist[1], _Pool_PSLindexToStructID ( $index_q1 ) )
_Pool_Container_CreateShare ( $IDlist[2], _Pool_PSLindexToStructID ( $index_q2 ) )
_Pool_Container_CreateShare ( $IDlist[3], _Pool_PSLindexToStructID ( $index_q3 ) )
_Pool_Container_CreateShare ( $IDlist[4], _Pool_PSLindexToStructID ( $index_q4 ) )

; create eight shares for the four input vectors (each is sent out to TWO Slaves)
_Pool_Container_CreateShare ( $IDlist[1], _Pool_PSLindexToStructID ( $index_left ) )
_Pool_Container_CreateShare ( $IDlist[3], _Pool_PSLindexToStructID ( $index_left ) )

_Pool_Container_CreateShare ( $IDlist[2], _Pool_PSLindexToStructID ( $index_right ) )
_Pool_Container_CreateShare ( $IDlist[4], _Pool_PSLindexToStructID ( $index_right ) )

_Pool_Container_CreateShare ( $IDlist[1], _Pool_PSLindexToStructID ( $index_top ) )
_Pool_Container_CreateShare ( $IDlist[2], _Pool_PSLindexToStructID ( $index_top ) )

_Pool_Container_CreateShare ( $IDlist[3], _Pool_PSLindexToStructID ( $index_bot ) )
_Pool_Container_CreateShare ( $IDlist[4], _Pool_PSLindexToStructID ( $index_bot ) )
```

Note that the last four input vectors are each Shared twice: $index_left with Slaves 1 and 3, $index_right with Slaves 2 and 4, $index_top with Slaves 1 and 2, and $index_bot with Slaves 3 and 4. This illustrates how a single physical Container can have multiple Shares that will be jointly controlled, by shipping them to all defined recipients at the same time. As explained earlier, this is done with Pool command "LoadSendContainer", which the Master script here sends to itself(!), by using **Pool** variable $_POOL_STATUS_ID as target. This **Pool** variable contains a process's own **Pool** ID, assigned by, and received from, the Post Office. We use the same trick as before to parse the Container IDs using Container indices.

```
_Pool_Send_Command ( $_POOL_STATUS_ID, "LoadSendContainer", _Pool_PSLindexToStructID ( $index_left ) )
_Pool_Send_Command ( $_POOL_STATUS_ID, "LoadSendContainer", _Pool_PSLindexToStructID ( $index_right ) )
_Pool_Send_Command ( $_POOL_STATUS_ID, "LoadSendContainer", _Pool_PSLindexToStructID ( $index_top ) )
_Pool_Send_Command ( $_POOL_STATUS_ID, "LoadSendContainer", _Pool_PSLindexToStructID ( $index_bot ) )
```

**Pool** command "LoadSendContainer" sets of a cascade of activity at both the Post Office, the Container's creator, and its recipient(s), involving the exchange of several messages about the identity and direction of transfer, and the locations in memory of data transit points. Unless something goes wrong, we do not care about such messages. This is followed by the actual sending of the Containers, which across a LAN would involve either TCP or UDP packets, but on a single machine simply means that the Post Office makes a copy of the sender's data in memory, and then proceeds to distribute it among all defined Sharing processes by telling each of *them* to make a copy of the Post Office's copy. Once all data are successfully transferred, the Post Office releases its temporary buffer to make space in memory for other transfers. This mechanism frees the Container's creator from all responsibilities of tracking transfer progress, resending missing or garbled parts, and dealing with various technical details. As soon as the local Post

Office has made its copy, the creator can move on to other matters, as will we.

While everyone else is frantically scurrying to get onto the same page(s), the Master thus has a few moments to prepare the next major step in getting this show on the road: writing the job descriptions for the Slaves. All **Pool** entities have the built-in ability to receive, store, and execute lines of **Autoit** script lines through their **Exe-Queue** (ExeQ). This encompasses both direct function calls and simple variable assignment (but not array cell assignment yet); basically, any **AutoIt** code that can be executed by a combination of Assign and Execute calls can be sent to any other **Pool** process for remote execution. For this example, we'll need to tell each Slave to perform three tasks:

1. map a matrix to each of the two input vectors, with **_Eigen_Add_ExistingMatrix()**
2. map a matrix to the (still empty) results matrix, also with **_Eigen_Add_ExistingMatrix()**
3. call **_Eigen_OuterProduct()** to fill the results matrix

This translates into four actual commands per Slave, to be stored in a temporary array called $jobs for clarity:

```
Global $jobs[5][5]      ; row = instruction ID, col = Slave ID, both base-1
For $cc = 1 To 4
        $jobs[0][$cc] = "Job for E4A Slave #" & $cc      ; for clarity only
Next
$jobs[0][0] = UBound ( $jobs ) - 1

; ExeQ cmds for Slave 1
$jobs[1][1] = _Pool_Container_ToE4AMatrix ( $index_left, "$matA" )
$jobs[2][1] = _Pool_Container_ToE4AMatrix ( $index_top, "$matB" )
$jobs[3][1] = _Pool_Container_ToE4AMatrix ( $index_q1, "$matR" )
$jobs[4][1] = "_Eigen_OuterProduct ( $matA, $matB, $matR )"

; ExeQ cmds for Slave 2
$jobs[1][2] = _Pool_Container_ToE4AMatrix ( $index_right, "$matA" )
$jobs[2][2] = _Pool_Container_ToE4AMatrix ( $index_top, "$matB" )
$jobs[3][2] = _Pool_Container_ToE4AMatrix ( $index_q2, "$matR" )
$jobs[4][2] = "_Eigen_OuterProduct ( $matA, $matB, $matR )"

; ExeQ cmds for Slave 3
$jobs[1][3] = _Pool_Container_ToE4AMatrix ( $index_left, "$matA" )
$jobs[2][3] = _Pool_Container_ToE4AMatrix ( $index_bot, "$matB" )
$jobs[3][3] =  _Pool_Container_ToE4AMatrix ( $index_q3, "$matR" )
$jobs[4][3] = "_Eigen_OuterProduct ( $matA, $matB, $matR )"

; ExeQ cmds for Slave 4
$jobs[1][4] =  _Pool_Container_ToE4AMatrix ( $index_right, "$matA" )
$jobs[2][4] = _Pool_Container_ToE4AMatrix ( $index_bot, "$matB" )
$jobs[3][4] = _Pool_Container_ToE4AMatrix ( $index_q4, "$matR" )        ; Hodor!
$jobs[4][4] = "_Eigen_OuterProduct ( $matA, $matB, $matR )"
```

We are using a special **Pool-E4A** hybrid function here: **_Pool_Container_ToE4Amatrix()**, which generates a tailored **_Eigen_Add_ExistingMatrix()** call that will work at the Container's destination. Note furthermore that each Slave uses the same variables ($matA, $matB, $matR) for different data (sub)sets; these do not clash, because the work is now split between different processes. So $matR of Slave 1 refers to quadrant1, $matR of Slave2 refers to quadrant2, and so on.

Once these batch jobs have been written, we still need to get them delivered to our worker bees. To that effect, we first fill a string variable called $batch with each line of the job, separated by carriage returns (@CR). Afterwards, we call Pool function **_Pool_Send_ExeQcall()**, which expects a Pool target ID and the

$batch variable as arguments. We perform this procedure four times to serve each of our four Slaves.

```
For $cc = 1 To 4
	$batch = $jobs[1][$cc]	; first line of batch job (ExeQ script)
	For $rc = 2 To $jobs[0][0]
		$batch &= @CR & $jobs[$rc][$cc]		; add next lines to execute
	Next

	; send batch job to specific Slave
	If _Pool_Send_ExeQcall ( $IDlist[$cc], $batch ) = False Then
		MsgBox ( 16, "Eigen4AutoIt HTC demo", "An error occurred while attempting to send a
batch job to Slave #" & $cc & ". Aborting..." )
		_Pool_Closedown ()
	EndIf
Next
```

There's two final steps before we can let this beast loose. We have to make sure that each Slave has received their batch job before we can tell them to execute it, and meanwhile we'll use two other **Pool** functions to set up automatic retrieval of the results as soon as each becomes available. Here's the code; the explanation follows.

```
_Pool_Send_Command ( $_POOL_BROADCAST, "ExeQReportState" )	; refresh ExeQ data (IPTR and
Queue size) in PML

$_POOL_STATUS_EXEQ_DONE_UDF = "_RetrieveResults"
$_POOL_STATUS_RECEIVED_UDF = "_ResultsReceived"
```

You'll have noticed we did not address each Slave individually here, but instead sent a single command with the special **Pool** ID variable $_POOL_BROADCAST. This is mainly for user convenience and script legibility; internally, this ID translates into a loop over all other Pool clients in the membership list, and individual messages are being sent to each. There are several other such subset identifiers; one can for instance target all Post Offices on a LAN, whereas others include or exclude all clients on a specific machine; check out the **Pool** documentation for more info.

The command we broadcast to all Slaves is "ExeQReportState", which triggers an immediate response in the form of a message detailing the total size of the Slave's ExeQ, and the current value of its instruction pointer. As each **Pool** process starts out with an empty ExeQ and we've sent four lines of code, we will be scanning for this increment. This is just the initial call; we will be repeating it in the next loop until all Slaves report having four instructions stored in their queue.

But before we get to that, we first define two functions to be called when certain events happen. A call to the first one. "_RetrieveResults", is triggered whenever a Slave reports that ExeQ execution is completed by sending an "EXEQ_DONE" message to its Master, which happens automatically. Since the Master is itself also partly event-driven, it will interrupt whatever it's doing whenever such a message comes in, to perfom the following operations:

```
Func _RetrieveResults ( $memberIndex )

	; check valid Pool member ID (NB PML index 1 = self, by definition)
	If $memberIndex < 2 Or $memberIndex > $_pool_MemberList[0][0] Then Return False

	; look up Slave
	Local $slaveID = _Pool_PMLindexToID ( $memberIndex )
	Local $index = _ArraySearch ( $IDlist, $slaveID, 1 )
	If $index < 1 Then Return False
```

```
        ; Instruct Slave to return the (filled) results Container to us
        _Pool_Send_Command ( $slaveID, "LoadSendContainer", _Pool_PSLindexToStructID ( $index ) )

        Return True
EndFunc
```

Notice anything special here? $_POOL_STATUS_EXEQ_DONE_UDF is not like Adlib-functions that accept no parameters; this particular event UDF always expects the Memberlist index of the **Pool** process that sent the message "EXEQ_DONE." In this particular case, the function converts that into a **Pool** ID (with **_Pool_PMLindexToID()**) and looks up that ID in its list of Slaves. Once it knows whether it is dealing with Slave number 1, 2, 3, or 4, the Master responds by sending Pool command "LoadSendContainer" to that slave, together with the Container ID of that specific Slave's results Container. So here is an example of a Container that is retrieved before its data were ever sent out. We only told the Slave to make space for it in memory (through the Share definition) and map a local matrix to it (though the ExeQ call). Now we want to get whatever the Slave stored in there, so we can stick that into our own matching matrix at our end.

The second function to be event-called, defined in $_POOL_STATUS_RECEIVED_UDF, is _ResultsReceived(). It is triggered whenever a Container shipment arrives, and any such function is expected to handle two parsed parameters: the sender's ID, and the Container ID) (a.k.a. structID). The only thing it does is to cross the Slave off the Master's list, so we can check elsewhere when all Slaves are done. Alternative functions could concievably also clean up the Container, or send it onward to another destination, which is why the Container ID is also parsed (although not used in this particular example).

```
Func _ResultsReceived ( $senderID, $structID )

        ; lookup Slave
        Local $index = _ArraySearch ( $IDlist, $senderID, 1 )
        If $index < 1 Then Return False

        $IDlist[$index] = False     ; done with you
        ConsoleWrite ( "_ResultsReceived from: " & $senderID & ", structID: " & Hex ( $structID, 8 ) &
@CRLF )

        Return True
EndFunc
```

We return to the main script, and to the next waiting loop, in which we keep broadcasting **Pool** command "ExeQReportState" until all four Slaves have received their four instructions to carry out. The only detail worth noting in the snippet below is the use of function **_Pool_getPMLcol()**, which yields a column index to the **Pool** members array (PML). **Pool**'s three most important arrays (PML, PSL, PSC) all work like this; they have a list of alphabetised column names associated with them, and this list is scanned to obtain the column where the requested variable is stored, just like a field in a database. As the PML contains over fifty variables per row, and new ones may be added in future, this provides flexibility, improves code legibility, and eliminates the possibility of index-based referencing errors. In this example, it is obvious that the evaluated PML fields concern the **Pool** member's ID and its current ExeQ size.

```
; wait for all Slaves to receive their ExeQ job
$tstart = TimerInit ()
While True
        $counter = 0
        For $cc = 2 To $_pool_MemberList[0][0]              ; skip index 1 = self
                If _ArraySearch ( $IDlist, $_pool_MemberList[$cc][ _Pool_getPMLcol ( "ID" ) ], 1 ) > 0 And _
```

```
                $_pool_MemberList[$cc][ _Pool_getPMLcol ( "ExeQ Size" ) ] = 4 Then $counter += 1
        Next
        If $counter = 4 Then ExitLoop     ; number of Slaves that received their batch job

        If TimerDiff ( $tstart ) > 30000 Then
                MsgBox ( 16, "Eigen4AutoIt HTC demo", "By now, the Slaves should have received their
batch jobs, so something went wrong. Aborting..." )
                _Pool_Closedown ()
        EndIf

        ; all Slaves should keep reporting changes in their ExeQueue
        _Pool_Send_Command ( $_POOL_BROADCAST, "ExeQReportState" )          ; refresh ExeQ data
(IPTR, Queue size) in PML
        Sleep ( 3000 )
WEnd
```

If you've come this far, you deserve to be rewarded:

*Thou shalt speak one power-word to All, and Lo, the Heavens and the Earth shall be moved by thy Minions, Oh Mighty One!*

Hyperbole aside, just broadcast the "Run" command, and all Slaves will start running their jobs for you. As we've just taken care of the automatic retrieval of each result as and when it becomes available, the only thing left for us to do is to check when all Slaves have been removed from our $IDlist. Subsequently, we can feel even better about ourselves by freeing our Slaves, who, without further purpose in life, then promptly kill themselves. All's well that ends well.

```
; Tell all Slaves to perform their assigned tasks
_Pool_Send_Command ( $_POOL_BROADCAST, "Run" )    ; Look! Look! See Slave Run!

; wait for Slaves to complete their job and return their results
$tstart = TimerInit ()
While True
        $results = 0
        For $cc = 1 To 4
                If $IDlist[$cc] = False Then $results += 1
        Next
        If $results = 4 Then ExitLoop

        If TimerDiff ( $tstart ) > 60 * 1000 Then
                MsgBox ( 16, "Eigen4AutoIt HTC demo", "By now, the four slaves should have finished
their jobs, so something went wrong. Aborting..." )
                _Pool_Closedown ()
        EndIf
        Sleep ( 250 )
WEnd

; free all Slaves
_Pool_Send_Command ( $_POOL_BROADCAST, "LeaveNow" )      ; sent to all other CLNs in Pool
```

We can now get rid of all Shares (with **_Pool_Container_WipePSC()**) and all Containers (with **_Pool_Container_Destroy_All()**). What then remains is to repeat the block-copy exercise to put each

quadrant in its proper place in our new, complete results matrix.

```
; reconstruct the full results matrix from its four parts
_Eigen_SetZero ( $matR )          ; just to make absolutely sure we're not cheating
_Eigen_Copy_Ablock_ToBblock ( $quadrant1, $matR, 0, 0, $halfsize, $halfsize, 0, 0 )
_Eigen_Copy_Ablock_ToBblock ( $quadrant2, $matR, 0, 0, $halfsize, $halfsize, 0, $halfsize )
_Eigen_Copy_Ablock_ToBblock ( $quadrant3, $matR, 0, 0, $halfsize, $halfsize, $halfsize, 0 )
_Eigen_Copy_Ablock_ToBblock ( $quadrant4, $matR, 0, 0, $halfsize, $halfsize, $halfsize, $halfsize )

; clean up our data transfer admin
_Pool_Container_WipePSC ()      ; remove all Share definitions for ALL Containers locally
_Pool_Container_Destroy_All ()  ; remove all shipping Containers
```

Obviously, this somewhat silly exercise was literally a waste of time, because (even if we ignore the time it took to set up the various actors on our stage) it took much longer to transfer the various bits and pieces forth and back than it did to do the job ourselves. But a cross-over point exists beyond which the work load is big enough (provided you have a sufficient number of idle CPU cores), to make multi-processing worth the extra effort. Plus always remember: **Pool** is Cool (please repeat five times before every meal).

Finally, is this multi-processed result actually identical to the one we produced ourselves? You'll find out when you run the Tutorial script, Question:  What is wrong with this last snippet below? Answer: It uses a MsgBox function to display the outcome of the evaluation, and MsgBox is a blocking function. We only get away with it here because we're basically done playing in the **Pool**. But you'd better not use such functions while you have data transfers in progress, or Slave processes doing your bidding. You may open a rift in the space-time continuum and end the universe as we know it.

The Tutorial script fizzles out by terminating the Post Office with extreme prejudice, before closing down its own **Pool** facilities. If **Pool** variable $_POOL_STATUS_STAYALIVE had been set to True, the script would continue afterwards, but our revels now are ended. So, good night unto you all.

```
$isEqual = _Eigen_IsCwiseEqual ( $matR, $matRlocal )
MsgBox ( 0, "Eigen4AutoIt HTC demo", "Are the two results equal: " & $isEqual )  ; this is BAD practice (blocking function)!

_MatrixDisplay ( $matR, "HTC result" )

; Shut it down!
ProcessClose ( $_POOL_ATTRIB_MGR_DEFAULT )          ; (c)rude, but effective
$_POOL_STATUS_STAYALIVE = False
_Pool_Closedown ()
```



## Eigen4AutoIt Function Reference

# Eigen4AutoIt Function Reference

Below is an alphabetical list of all main functions available in **Eigen4AutoIt**. Click on a function name for a detailed description.

See *Function Notes* for general remarks on usage, and *Complex Function Notes* for complex usage.

| Eigen4AutoIt Function | Description |
|---|---|
| _Eigen_Add_ExistingMatrix | Add an externally-created matrix struct to the MatrixList |
| _Eigen_Adjoint | Complex equivalent of transposition (see _Eigen_Transpose) |
| _Eigen_Adjoint_InPlace | Complex equivalent of transposition (see _Eigen_Transpose_InPlace); stored in-place |
| _Eigen_Center | Cellwise-subtract the global matrix mean, so all values center around zero |
| _Eigen_Center_Colwise | Cellwise-subtract the mean per column, so all values center around zero per column |
| _Eigen_Center_Colwise_InPlace | Cellwise-subtract the mean per column, so all values center around zero per column; stored in-place |
| _Eigen_Center_InPlace | Cellwise-subtract the global matrix mean, so all values center around zero |
| _Eigen_Center_Rowwise | Cellwise-subtract the mean per row, so all values center around zero per row |
| _Eigen_Center_Rowwise_InPlace | Cellwise-subtract the mean per row, so all values center around zero per row; stored in-place |
| _Eigen_Cleanup | Cleans up an Eigen4AutoIt work environment initialised with _Eigen_StartUp |
| _Eigen_CloneArray | Duplicate an existing array, and optionally copy its data too |
| _Eigen_CloneMatrix | Duplicate an existing matrix, and optionally copy its data too |
| _Eigen_ConditAll | Evaluate whether *all* cells of matrix A satisfy [condition, scalar] |
| _Eigen_ConditAll_Col | Evaluate whether *all* cells of a column in matrix A satisfy [condition, scalar] |
| _Eigen_ConditAll_Row | Evaluate whether *all* cells of a row in matrix A satisfy [condition, scalar] |
| _Eigen_ConditAny | Evaluate whether *any* cell of matrix A satisfies [condition, scalar] |
| _Eigen_ConditAny_Col | Evaluate whether *any* cell of a specified column in matrix A satisfies [condition, scalar] |
| _Eigen_ConditAny_Row | Evaluate whether *any* cell of a specified row in matrix A satisfies [condition, scalar] |
| _Eigen_ConditBinaryOp | Apply [operator, value] to each cell of matrix A that satisfies [condition, scalar], using the value in the corresponding cell in matrix B |
| _Eigen_ConditBinaryOp_InPlace | Apply [operator, value] to each cell of matrix A that satisfies [condition, scalar], using the value in the corresponding cell in matrix B, storing the result in-place |
| _Eigen_ConditCount | Evaluate *how many* cells of matrix A satisfy [condition, scalar] |
| _Eigen_ConditCount_Col | Evaluate *how many* cells of a specified column in matrix A satisfy [condition, scalar] |
| _Eigen_ConditCount_Row | Evaluate *how many* cells of a specified row in matrix A satisfy [condition, scalar] |

| | |
|---|---|
| _Eigen_ConditScalarOp | Apply [operator, scalar] to each cell of matrix A that satisfies [condition, scalar] |
| _Eigen_ConditScalarOp_InPlace | Apply [operator, scalar] to each cell of matrix A that satisfies [condition, scalar], storing the result in-place |
| _Eigen_ConditUnaryOp | Apply [operator] to each cell of matrix A that satisfies [condition, scalar] |
| _Eigen_ConditUnaryOp_InPlace | Apply [operator] to each cell of matrix A that satisfies [condition, scalar], storing the result in-place |
| _Eigen_ConvertMatrix_ToComplexDouble | Convert a matrix of any type to type "complex double" in memory |
| _Eigen_ConvertMatrix_ToComplexFloat | Convert a matrix of any type to type "complex float" in memory |
| _Eigen_ConvertMatrix_ToDouble | Convert a matrix of any type to type "real double" in memory |
| _Eigen_ConvertMatrix_ToFloat | Convert a matrix of any type to type "real float" in memory |
| _Eigen_ConvertMatrix_ToInt | Convert a matrix of any type to type "int" in memory |
| _Eigen_ConvertMatrixFile_ToComplexDouble | Convert a matrix file of any type to type "complex double" |
| _Eigen_ConvertMatrixFile_ToComplexFloat | Convert a matrix file of any type to type "complex float" |
| _Eigen_ConvertMatrixFile_ToDouble | Convert a matrix file of any type to type "real double" |
| _Eigen_ConvertMatrixFile_ToFloat | Convert a matrix file of any type to type "real float" |
| _Eigen_ConvertMatrixFile_ToInt | Convert a matrix file of any type to type "int" |
| _Eigen_Copy_A_ToB | Copy all data from matrix A to (existing) matrix B |
| _Eigen_Copy_A_ToBimag | Copy all data from real matrix A to (existing) complex matrix B's imaginary part |
| _Eigen_Copy_A_ToBreal | Copy all data from real matrix A to (existing) complex matrix B's real part |
| _Eigen_Copy_Ablock_ToAblock | Copy all data from a block in matrix A to another block in matrix A |
| _Eigen_Copy_Ablock_ToBblock | Copy all data from a block in matrix A to a block in matrix B |
| _Eigen_Copy_Acol_ToBcol | Copy all data from a column in matrix A to a column in matrix B |
| _Eigen_Copy_Acol_ToBdiag | Copy all data from a column in matrix A to the diagonal of matrix B |
| _Eigen_Copy_Acol_ToBrow | Copy all data from a column in matrix A to a row in matrix B |
| _Eigen_Copy_Adiag_ToBcol | Copy all data from the diagonal of matrix A to a column of matrix B |
| _Eigen_Copy_Adiag_ToBdiag | Copy all data from the diagonal of matrix A to the diagonal of matrix B |
| _Eigen_Copy_Adiag_ToBrow | Copy all data from the diagonal of matrix A to a row of matrix B |
| _Eigen_Copy_Adiag_ToBvector | Copy all data from the diagonal of matrix A to vector B |
| _Eigen_Copy_Aimag_ToB | Copy all data from complex matrix A's imaginary part to (existing) real matrix B |
| _Eigen_Copy_Aimag_ToBimag | Copy all data from complex matrix A's imaginary part to |

| | |
|---|---|
| | (existing) complex matrix B's imaginary part |
| _Eigen_Copy_Aimag_ToBreal | Copy all data from complex matrix A's imaginary part to (existing) complex matrix B's real part |
| _Eigen_Copy_ALower_ToBLower | Copy all data from the lower triangular of matrix A to the lower triangular of matrix B |
| _Eigen_Copy_ALower_ToBUpper | Copy all data from the lower triangular of matrix A to the upper triangular of matrix B |
| _Eigen_Copy_Areal_ToB | Copy all data from complex matrix A's real part to (existing) real matrix B |
| _Eigen_Copy_Areal_ToBimag | Copy all data from complex matrix A's real part to (existing) complex matrix B's imaginary part |
| _Eigen_Copy_Areal_ToBreal | Copy all data from complex matrix A's real part to (existing) complex matrix B's real part |
| _Eigen_Copy_Arow_ToBcol | Copy all data from a row of matrix A to a column of matrix B |
| _Eigen_Copy_Arow_ToBdiag | Copy all data from a row in matrix A to the diagonal of matrix B |
| _Eigen_Copy_Arow_ToBrow | Copy all data from a row of matrix A to a row of matrix B |
| _Eigen_Copy_ArrayData_ToMatrix | Copy all array data to a same-sized, existing matrix |
| _Eigen_Copy_AStrictlyLower_ToBLower | Copy all data below the diagonal of matrix A to the same area in matrix B |
| _Eigen_Copy_AStrictlyUpper_ToBUpper | Copy all data above the diagonal of matrix A to the same area in matrix B |
| _Eigen_Copy_AUnitLower_ToBLower | Copy all data below the diagonal in matrix A to the same area in matrix B, and fill the latter's diagonal with ones |
| _Eigen_Copy_AUnitUpper_ToBUpper | Copy all data above the diagonal in matrix A to the same area in matrix B, and fills the latter's diagonal with ones |
| _Eigen_Copy_AUpper_ToBLower | Copy all data from the upper triangular of matrix A to the lower triangular of matrix B |
| _Eigen_Copy_AUpper_ToBUpper | Copy all data from the upper triangular of matrix A to the upper triangular of matrix B |
| _Eigen_Copy_Avector_ToBdiag | Copy all data from vector A to the diagonal of matrix B |
| _Eigen_Copy_MatrixData_ToArray | Copy all matrix data to a same-sized, existing array |
| _Eigen_Correlation_AB | Compute the correlation coefficient for two same-sized matrices |
| _Eigen_Correlation_ColCol | Compute the correlation coefficient for two columns in a matrix |
| _Eigen_Correlation_RowRow | Compute the correlation coefficient for two rows in a matrix |
| _Eigen_Covariance_Colwise | Compute the covariance matrix (standard form) |
| _Eigen_Covariance_Rowwise | Compute the covariance matrix (non-standard, transposed form) |
| _Eigen_CreateArray_FromMatrix | Create a new array from an existing matrix, and optionally copy all matrix data too |
| _Eigen_CreateMatrix | Allocate a new matrix variable of specified dimensions in memory |
| _Eigen_CreateMatrix_Constant | Allocate a new matrix variable of specified dimensions in memory, fill all cells with supplied value |
| _Eigen_CreateMatrix_FromA | Create a matrix from multiple copies of the source matrix values, tiled horizontally, vertically, or both |

| | |
|---|---|
| _Eigen_CreateMatrix_FromA_Transposed | Create a matrix from multiple copies of the transposed source matrix values, tiled horizontally, vertically, or both |
| _Eigen_CreateMatrix_FromABcols | Create a new matrix from all columns of matrix A, followed by all columns of matrix B |
| _Eigen_CreateMatrix_FromABcols_Transposed | Create a new matrix from all columns of matrix A, followed by all columns of matrix B, transposed |
| _Eigen_CreateMatrix_FromAblock | Create a new matrix from an existing block in matrix A, including its contents |
| _Eigen_CreateMatrix_FromAblock_Transposed | Create a new matrix from an existing block in matrix A, including its contents, transposed |
| _Eigen_CreateMatrix_FromABrows | Create a new matrix from all rows of matrix A, followed by all rows of matrix B |
| _Eigen_CreateMatrix_FromABrows_Transposed | Create a new matrix from all rows of matrix A, followed by all rows of matrix B, transposed |
| _Eigen_CreateMatrix_FromAcols | Create a new matrix from listed columns of matrix A |
| _Eigen_CreateMatrix_FromAcols_Transposed | Create a new matrix from listed columns of matrix A, transposed |
| _Eigen_CreateMatrix_FromArows | Create a new matrix from listed rows of matrix A |
| _Eigen_CreateMatrix_FromArows_Transposed | Create a new matrix from listed rows of matrix A, transposed |
| _Eigen_CreateMatrix_FromArray | Create a new matrix from an existing array, and optionally copy all array data too |
| _Eigen_CreateMatrix_Identity | Allocate a new matrix variable of specified dimensions in memory, zero all cells, fill diagonal with ones |
| _Eigen_CreateMatrix_LinSpaced_ColMajor | Allocate a new matrix variable of specified dimensions in memory, fill all cells in ColMajor order with succession of equally-spaced values |
| _Eigen_CreateMatrix_LinSpaced_RowMajor | Allocate a new matrix variable of specified dimensions in memory, fill all cells in RowMajor order with succession of equally-spaced values |
| _Eigen_CreateMatrix_Ones | Allocate a new matrix variable of specified dimensions in memory, fill all cells with ones |
| _Eigen_CreateMatrix_Random | Allocate a new matrix variable of specified dimensions in memory, fill all cells with random values (0-1) |
| _Eigen_CreateMatrix_Zero | Allocate a new matrix variable of specified dimensions in memory, fill all cells with zeroes |
| _Eigen_CreateOutput_FromDims | Allocate a new output matrix variable for a given function and input matrix dimensions |
| _Eigen_CreateOutput_FromIDs | Allocate a new output matrix variable for a given function and input matrix IDs |
| _Eigen_CrossProduct | Compute the cross-product of two same-sized vectors |
| _Eigen_CwiseBinaryOp | Apply [operator, value] to each cell of matrix A, using the value in the corresponding cell in matrix B |
| _Eigen_CwiseBinaryOp_Block | Apply [operator, value] to each cell of a block in matrix A, using the value in the corresponding cell in a same-sized block in |

| | |
|---|---|
| | matrix B |
| _Eigen_CwiseBinaryOp_Block_InPlace | Apply [operator, value] to each cell of a block in matrix A, using the value in the corresponding cell in a same-sized block in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_Col | Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in matrix B |
| _Eigen_CwiseBinaryOp_Col_InPlace | Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_ColCol | Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in another column in matrix B |
| _Eigen_CwiseBinaryOp_ColCol_InPlace | Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in another column in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_ColRow | Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in a column in matrix B |
| _Eigen_CwiseBinaryOp_ColRow_InPlace | Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in a column in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_Colwise | Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in Colvector B |
| _Eigen_CwiseBinaryOp_Colwise_InPlace | Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in Colvector B, storing the result in-place |
| _Eigen_CwiseBinaryOp_ColwiseCol | Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in a single column in matrix B |
| _Eigen_CwiseBinaryOp_ColwiseCol_InPlace | Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in a single column in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_InPlace | Apply [operator, value] to each cell of matrix A, using the value in the corresponding cell in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_Row | Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in matrix B |
| _Eigen_CwiseBinaryOp_Row_InPlace | Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_RowCol | Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in one row in matrix B |
| _Eigen_CwiseBinaryOp_RowCol_InPlace | Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in one row in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_RowRow | Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in a row in matrix B |
| _Eigen_CwiseBinaryOp_RowRow_InPlac | Apply [operator, value] to each cell of one row in matrix A, using |

| | |
|---|---|
| e | the value in the corresponding cell in a row in matrix B, storing the result in-place |
| _Eigen_CwiseBinaryOp_Rowwise | Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in Rowvector B |
| _Eigen_CwiseBinaryOp_Rowwise_InPlace | Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in Rowvector B, storing the result in-place |
| _Eigen_CwiseBinaryOp_RowwiseRow | Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in a single row in matrix B |
| _Eigen_CwiseBinaryOp_RowwiseRow_InPlace | Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in a single row in matrix B, storing the result in-place |
| _Eigen_CwiseScalarOp | Apply [operator, scalar] to each cell of matrix A |
| _Eigen_CwiseScalarOp_Block | Apply [operator, scalar] to each cell of a block in matrix A |
| _Eigen_CwiseScalarOp_Block_InPlace | Apply [operator, scalar] to each cell of a block in matrix A, storing the result in-place |
| _Eigen_CwiseScalarOp_Col | Apply [operator, scalar] to each cell of one column in matrix A |
| _Eigen_CwiseScalarOp_Col_InPlace | Apply [operator, scalar] to each cell of one column in matrix A, storing the result in-place |
| _Eigen_CwiseScalarOp_ColCol | Apply [operator, scalar] to each cell of one column in matrix A, storing the result in another column in results matrix R |
| _Eigen_CwiseScalarOp_ColCol_InPlace | Apply [operator, scalar] to each cell of one column in matrix A, storing the result in another column in matrix A |
| _Eigen_CwiseScalarOp_ColRow | Apply [operator, scalar] to each cell of one column in matrix A, storing the result in a row in results matrix R |
| _Eigen_CwiseScalarOp_ColRow_InPlace | Apply [operator, scalar] to each cell of one column in matrix A, storing the result in a row in matrix A |
| _Eigen_CwiseScalarOp_InPlace | Apply [operator, scalar] to each cell of matrix A, storing the result in-place |
| _Eigen_CwiseScalarOp_Row | Apply [operator, scalar] to each cell of one row in matrix A |
| _Eigen_CwiseScalarOp_Row_InPlace | Apply [operator, scalar] to each cell of one row in matrix A, storing the result in-place |
| _Eigen_CwiseScalarOp_RowCol | Apply [operator, scalar] to each cell of one row in matrix A, storing the result in a column in results matrix R |
| _Eigen_CwiseScalarOp_RowCol_InPlace | Apply [operator, scalar] to each cell of one row in matrix A, storing the result in a column in matrix A |
| _Eigen_CwiseScalarOp_RowRow | Apply [operator, scalar] to each cell of one row in matrix A, storing the result in another row in results matrix R |
| _Eigen_CwiseScalarOp_RowRow_InPlace | Apply [operator, scalar] to each cell of one row in matrix A, storing the result in another row in matrix A |
| _Eigen_CwiseUnaryOp | Apply [operator] to each cell of matrix A |
| _Eigen_CwiseUnaryOp_Block | Apply [operator] to each cell of a block in matrix A |
| _Eigen_CwiseUnaryOp_Block_InPlace | Apply [operator] to each cell of a block in matrix A, storing the |

| | |
|---|---|
| | result in-place |
| _Eigen_CwiseUnaryOp_Col | Apply [operator] to each cell of one column in matrix A |
| _Eigen_CwiseUnaryOp_Col_InPlace | Apply [operator] to each cell of one column in matrix A, storing the result in-place |
| _Eigen_CwiseUnaryOp_ColCol | Apply [operator] to each cell of a column in matrix A, storing the result in another column in results matrix R |
| _Eigen_CwiseUnaryOp_ColCol_InPlace | Apply [operator] to each cell of a column in matrix A, storing the result in another column in matrix A |
| _Eigen_CwiseUnaryOp_ColRow | Apply [operator] to each cell of one column in matrix A, storing the result in a row in rsults matrix R |
| _Eigen_CwiseUnaryOp_ColRow_InPlace | Apply [operator] to each cell of one column in matrix A, storing the result in a row in matrix A |
| _Eigen_CwiseUnaryOp_InPlace | Apply [operator] to each cell of matrix A, storing the result in-place |
| _Eigen_CwiseUnaryOp_Row | Apply [operator] to each cell of one row in matrix A |
| _Eigen_CwiseUnaryOp_Row_InPlace | Apply [operator] to each cell of one row in matrix A, storing the result in-place |
| _Eigen_CwiseUnaryOp_RowCol | Apply [operator] to each cell of one row in matrix A, storing the result in a column in results matrix R |
| _Eigen_CwiseUnaryOp_RowCol_InPlace | Apply [operator] to each cell of one row in matrix A, storing the result in a column in matrix A |
| _Eigen_CwiseUnaryOp_RowRow | Apply [operator] to each cell of one row in matrix A, storing the result in another row in results matrix R |
| _Eigen_CwiseUnaryOp_RowRow_InPlace | Apply [operator] to each cell of one row in matrix A, storing the result in another row in matrix A |
| _Eigen_DebugMode_Off | Switch from Debug mode to Computing mode |
| _Eigen_DebugMode_On | Switch from Computing mode to Debug mode |
| _Eigen_Decomp_Choleski | Perform a Choleski decomposition |
| _Eigen_Decomp_ComplexSchur | Perform a ComplexSchur helper decomposition |
| _Eigen_Decomp_Hessenberg | Perform a Hessenberg helper decomposition |
| _Eigen_Decomp_HouseholderQR | Perform a HouseholderQR decomposition |
| _Eigen_Decomp_JacobiSVD | Perform a JacobiSVD decomposition |
| _Eigen_Decomp_LowerUpper | Perform a LowerUpper (LU) decomposition |
| _Eigen_Decomp_RealQZ | Perform a RealQZ helper decomposition |
| _Eigen_Decomp_RealSchur | Perform a RealSchur helper decomposition |
| _Eigen_Decomp_Tridiagonalization | Perform a Tridiagonalization decomposition |
| _Eigen_DotProduct | Compute the dot-product of two same-sized vectors |
| _Eigen_EigenSolver | Perform eigen analysis on a real input matrix |
| _Eigen_EigenSolver_Complex | Perform eigen analysis on a complex input matrix |
| _Eigen_EigenSolver_Generalized | Perform generalized eigen analysis on two input matrices |
| _Eigen_EigenSolver_Generalized_SelfAdjoint | Perform generalized eigen analysis on a symmetric matrix |

| | |
|---|---|
| _Eigen_EigenSolver_SelfAdjoint | Perform eigen analysis on a symmetric matrix |
| _Eigen_GetActiveMatrix | Retrieve the matrix ID for a generic matrix output, identified by letter (A-Z) |
| _Eigen_GetMatrixCols | Return matrix column dimension |
| _Eigen_GetMatrixRows | Return matrix row dimension |
| _Eigen_GetMatrixType | Return matrix variable type as string |
| _Eigen_GetMatrixTypeID | Return matrix variable type as integer ID |
| _Eigen_Hide_Errors | Disable all non-fatal error messages |
| _Eigen_Hide_Warnings | Disable Warning-Proceed? dialogs in unexpected situations |
| _Eigen_Inverse | Compute the matrix inverse |
| _Eigen_Inverse_InPlace | Compute the matrix inverse; stored in-place |
| _Eigen_IsActiveMatrix | Retrieve the active/inactive state for a generic matrix output, identified by letter (A-Z) |
| _Eigen_IsComplex | Return True if the type of a parsed matrix ID is complex float or complex double |
| _Eigen_IsCwiseEqual_AB | Evaluate whether input matrices A and B are equal, to within some small margin |
| _Eigen_IsCwiseEqual_ToConstant | Evaluate whether all cells of input matrix A are equal to a constant, to within some small margin |
| _Eigen_Kurtosis | Return the global matrix kurtosis (peakedness) |
| _Eigen_Kurtosis_Col | Return the kurtosis (peakedness) for a specified matrix column |
| _Eigen_Kurtosis_Row | Return the kurtosis (peakedness) for a specified matrix row |
| _Eigen_LeastSquares | Solve a sytem of linear equations in the least-squares sense |
| _Eigen_LoadMatrix | Load a new matrix variable from file |
| _Eigen_MatrixDisplay | Display matrix in a ListView |
| _Eigen_MatrixSpecs | Evaluate and return a range of Matrix Specs for a matrix |
| _Eigen_MatrixSpecs_Block | Evaluate and return a range of Matrix Specs for a block |
| _Eigen_MatrixSpecs_Block_Single | Evaluate and return a single Matrix Spec for a defined block |
| _Eigen_MatrixSpecs_Col | Evaluate and return a range of Matrix Specs for one column |
| _Eigen_MatrixSpecs_Col_Single | Evaluate and return a single Matrix Spec for one column |
| _Eigen_MatrixSpecs_Colwise_Single | Evaluate and return a specified Matrix Spec for each column; store the result in a Rowvector |
| _Eigen_MatrixSpecs_Diag | Evaluate and return a range of Matrix Specs for the matrix diagonal |
| _Eigen_MatrixSpecs_Diag_Single | Evaluate and return a single Matrix Spec for the matrix diagonal |
| _Eigen_MatrixSpecs_Row | Evaluate and return a range of Matrix Specs for one selected row |
| _Eigen_MatrixSpecs_Row_Single | Evaluate and return a single Matrix Spec for one selected row |
| _Eigen_MatrixSpecs_Rowwise_Single | Evaluate and return a specified Matrix Spec for each row; store the result in a Colvector |
| _Eigen_MatrixSpecs_Single | Evaluate and return one specifiied Matrix Spec for a matrix |

| | |
|---|---|
| _Eigen_Misfit | Compute the sum of *absolute* residuals after obtaining a model fit |
| _Eigen_Misfit_Col | Compute the sum of *absolute* residuals after obtaining a model fit, for a specified column of observations matrix B |
| _Eigen_Misfit_Colwise | Compute the sum of *absolute* residuals after obtaining a model fit, per column of observations matrix B |
| _Eigen_Multiply_A_BdivC | Multiply input matrix A with the result of Cwise-dividing matrix B by C |
| _Eigen_Multiply_A_BminusC | Multiply input matrix A with the result of Cwise-subtracting matrix C from matrix B |
| _Eigen_Multiply_A_BplusC | Multiply input matrix A with the result of Cwise-adding matrices B and C |
| _Eigen_Multiply_AA | Multiply input matrix A with itself |
| _Eigen_Multiply_AA_InPlace | Multiply input matrix A with itself, and store the result in matrix A |
| _Eigen_Multiply_AAt | Multiply input matrix A with itself transposed |
| _Eigen_Multiply_AAt_InPlace | Multiply input matrix A with itself transposed, and store the result in matrix A |
| _Eigen_Multiply_AB | Multiply input matrix A with input matrix B |
| _Eigen_Multiply_AB_divC | Multiply input matrices A and B first, and then Cwise-divide by the contents of matrix C |
| _Eigen_Multiply_AB_InPlace | Multiply input matrix A with input matrix B, and store the result in matrix A |
| _Eigen_Multiply_AB_minusC | Multiply input matrices A and B first, and then Cwise-subtract the contents of matrix C |
| _Eigen_Multiply_AB_plusC | Multiply input matrices A and B first, and then Cwise-add the contents of matrix C |
| _Eigen_Multiply_ABC | Multiply input matrices A, B, and C |
| _Eigen_Multiply_ABC_InPlace | Multiply input matrices A, B, and C, and store the result in matrix A |
| _Eigen_Multiply_ABCD | Multiply input matrices A, B, C, and D |
| _Eigen_Multiply_ABCD_InPlace | Multiply input matrices A, B, C, and D, and store the result in matrix A |
| _Eigen_Multiply_ABCDE | Multiply input matrices A, B, C, D, and E |
| _Eigen_Multiply_ABCDE_InPlace | Multiply input matrices A, B, C, D, and E, and store the result in matrix A |
| _Eigen_Multiply_ABt | Multiply input matrix A with input matrix B transposed |
| _Eigen_Multiply_ABt_InPlace | Multiply input matrix A with input matrix B transposed, and store the result in matrix A |
| _Eigen_Multiply_AtA | Multiply input matrix A transposed with itself |
| _Eigen_Multiply_AtA_InPlace | Multiply input matrix A transposed with itself, and store the result in matrix A |
| _Eigen_Multiply_AtAt | Multiply input matrix A transposed with itself transposed |
| _Eigen_Multiply_AtAt_InPlace | Multiply input matrix A transposed with itself transposed, and store the result in matrix A |

| | |
|---|---|
| _Eigen_Multiply_AtB | Multiply input matrix A transposed with input matrix B |
| _Eigen_Multiply_AtB_InPlace | Multiply input matrix A transposed with input matrix B, and store the result in matrix A |
| _Eigen_Multiply_AtBt | Multiply input matrix A transposed with input matrix B transposed |
| _Eigen_Multiply_AtBt_InPlace | Multiply input matrix A transposed with input matrix B transposed, and store the result in matrix A |
| _Eigen_Normalise | Cellwise-divide by the global matrix standard deviation |
| _Eigen_Normalise_Colwise | Cellwise-divide by the standard deviation per column, so all values are normalised per column |
| _Eigen_Normalise_Colwise_InPlace | Cellwise-divide by the standard deviation per column, so all values are normalised per column; store in-place |
| _Eigen_Normalise_InPlace | Cellwise-divide by the global matrix standard deviation; store in-place |
| _Eigen_Normalise_Rowwise | Cellwise-divide by the standard deviation per row, so all values are normalised per row |
| _Eigen_Normalise_Rowwise_InPlace | Cellwise-divide by the standard deviation per row, so all values are normalised per row; store in-place |
| _Eigen_OuterProduct | Multiply a Rowvector with a Colvector to create a (multiplication table) matrix of their joint dimensions |
| _Eigen_PartialLeastSquares | Solve a sytem of linear equations in the least-squares sense in eigenspace |
| _Eigen_PCA | Perform a Principal Components Analysis (PCA) |
| _Eigen_PCA_ReDim | Perform dimensional compression after Principal Components Analysis (PCA) |
| _Eigen_PseudoInverse | Compute the Penrose-Moore pseudo-inverse using Courrieu's algorithm |
| _Eigen_ReadMatrixValue | Read the contents of a specific matrix cell |
| _Eigen_RedefineMatrix | Re-allocate an existing matrix of specified, new dimensions in memory |
| _Eigen_Redim_ExistingMatrix | Reorganise the dimensionality of an existing matrix without altering its content |
| _Eigen_Redim_ExistingMatrixFile | Reorganise the dimensionality of a matrix stored on file, without altering its content |
| _Eigen_RelativeError | Compute the ratio of a model fit's residual norm over the observational norm |
| _Eigen_RelativeError_Col | Compute the ratio of a model fit's residual norm over the observational norm, for a specified column of observations matrix B |
| _Eigen_RelativeError_Colwise | Compute the ratio of a model fit's residual norm over the observational norm, per column of observations matrix B |
| _Eigen_ReleaseFromIndex | Release the indexed matrix from memory, as well as all matrices created thereafter |
| _Eigen_ReleaseFromIndex_Except | Release the indexed matrix from memory, as well as all matrices created thereafter, with exceptions |

| | |
|---|---|
| _Eigen_ReleaseFromIndex_ToIndex | Release from memory all matrices within the specified index range |
| _Eigen_ReleaseFromIndex_ToIndex_Except | Release from memory all matrices within the specified index range, with exceptions |
| _Eigen_ReleaseFromMarker | Release from memory all matrices created after the ListMarker was set |
| _Eigen_ReleaseFromMarker_Except | Release from memory all matrices created after the ListMarker was set, with exceptions |
| _Eigen_ReleaseMatrix | Release an existing matrix from memory and from the MatrixList |
| _Eigen_ReleaseMatrix_All | Release all existing matrices from memory and reset the MatrixList to zero |
| _Eigen_ReleaseMatrix_All_Except | Release all existing matrices from memory, with exceptions |
| _Eigen_ResetActiveMatrix | Reset all generic output matrices as inactive |
| _Eigen_ResetActiveMatrix_Single | Reset selected output matrices as inactive |
| _Eigen_ResetListMarker | Clear the ListMarker |
| _Eigen_Reverse | Mirror-exchange the cell contents of a matrix horizontally and vertically |
| _Eigen_Reverse_Block | Mirror-exchange a selected block's cell contents horizontally and vertically |
| _Eigen_Reverse_Block_InPlace | Mirror-exchange a selected block's cell contents horizontally and vertically; stored in-place |
| _Eigen_Reverse_Col | Mirror-exchange the cell contents of a selected column |
| _Eigen_Reverse_Col_InPlace | Mirror-exchange the cell contents of a selected column; stored in-place |
| _Eigen_Reverse_Diag | Mirror-exchange the cell contents of the diagonal |
| _Eigen_Reverse_Diag_InPlace | Mirror-exchange the cell contents of the diagonal; stored in-place |
| _Eigen_Reverse_InPlace | Mirror-exchange the cell contents of a matrix horizontally and vertically; stored in-place |
| _Eigen_Reverse_Row | Mirror-exchange the cell contents of a selected row |
| _Eigen_Reverse_Row_InPlace | Mirror-exchange the cell contents of a selected row; stored in-place |
| _Eigen_RsqAdjusted | Compute the percentage of total variability explained by a model fit, adjusted for overfitting |
| _Eigen_Rsquared | Compute the percentage of total variability explained by a model fit |
| _Eigen_Rsquared_Col | Compute the percentage of total variability explained by a model fit, for a specified column of observations matrix B |
| _Eigen_Rsquared_Colwise | Compute the percentage of total variability explained by a model fit, per column of observations matrix B |
| _Eigen_SaveMatrix | Save an existing matrix variable to file |
| _Eigen_SetActiveMatrix | Flag one or more generic matrix outputs (identified by letter) as active, i.e., to create and/or fill with data |

| | |
|---|---|
| _Eigen_SetConstant | Fill all matrix cells with the same value |
| _Eigen_SetConstant_Block | Fill all matrix cells in a block with the same value |
| _Eigen_SetConstant_Col | Fill all matrix cells in one column with the same value |
| _Eigen_SetConstant_Diag | Fill all diagonal cells of a matrix with the same value |
| _Eigen_SetConstant_Row | Fill all matrix cells in one row with the same value |
| _Eigen_SetIdentity | Fill a matrix with zeroes, and the diagonal with ones |
| _Eigen_SetIdentity_Block | Fill a matrix block with zeroes, and the block diagonal with ones |
| _Eigen_SetLinSpaced_Col | Fill all matrix cells in one column with a succession of equally-spaced values |
| _Eigen_SetLinSpaced_ColMajor | Fill all matrix cells in ColMajor order with a succession of equally-spaced values |
| _Eigen_SetLinSpaced_Diag | Fill all diagonal cells of a matrix with a succession of equally-spaced values |
| _Eigen_SetLinSpaced_Row | Fill all matrix cells in one row with a succession of equally-spaced values |
| _Eigen_SetLinSpaced_RowMajor | Fill all matrix cells in RowMajor order with a succession of equally-spaced values |
| _Eigen_SetListMarker | Store the current-last index of the MatrixList for future reference |
| _Eigen_SetMatrixType | Select a matrix variable type prior to a calling _Eigen_Startup() |
| _Eigen_SetOnes | Fill all matrix cells with ones |
| _Eigen_SetOnes_Block | Fill all matrix cells in a block with ones |
| _Eigen_SetOnes_Col | Fill all matrix cells in one column with ones |
| _Eigen_SetOnes_Diag | Fill all diagonal cells of a matrix with ones |
| _Eigen_SetOnes_Row | Fill all matrix cells in a specific row with ones |
| _Eigen_SetRandom | Fill all matrix cells with random values in the range 0-1 |
| _Eigen_SetRandom_Block | Fill all matrix cells in a block with random values in the range 0-1 |
| _Eigen_SetRandom_Col | Fill all matrix cells in a specific column with random values in the range 0-1 |
| _Eigen_SetRandom_Diag | Fill all diagonal cells of a matrix with random values in the range 0-1 |
| _Eigen_SetRandom_Row | Fill all matrix cells in one row with random values in the range 0-1 |
| _Eigen_SetZero | Fill all matrix cells with zero |
| _Eigen_SetZero_Block | Fill all matrix cells in a block with zero |
| _Eigen_SetZero_Col | Fill all matrix cells in one column with zero |
| _Eigen_SetZero_Diag | Fill all diagonal cells of a matrix with zero |
| _Eigen_SetZero_Row | Fill all matrix cells in one row with zero |
| _Eigen_Show_AllFunctions | Display a list of **Eigen4AutoIt**'s functions, with brief description |
| _Eigen_Show_BinaryOperators | Display the list of Cwise binary operators |
| _Eigen_Show_ConditOperators | Display the list of Cwise conditional operators |
| _Eigen_Show_Constants_Nist | Display a list of **Eigen4AutoIt**'s physics constants, compiled by NIST |

| | |
|---|---|
| _Eigen_Show_Constants_Sykora1 | Display a list of **Eigen4AutoIt**'s mathematical constants, compiled by Sykora |
| _Eigen_Show_Constants_Sykora2 | Display a list of **Eigen4AutoIt**'s scientific constants, compiled by Sykora |
| _Eigen_Show_DecompSpecs | Display the list of decomposition Specs variables |
| _Eigen_Show_DecompSpecs_Current | Display the current contents of the decomposition Specs |
| _Eigen_Show_DllFunctions | Display a list of **Eigen4AutoIt**'s supported matrixtype-specific function variants |
| _Eigen_Show_EnvironmentVars | Display a list of **Eigen4AutoIt**'s current environment settings |
| _Eigen_Show_Errors | Display all non-fatal error messages |
| _Eigen_Show_MatrixList | Display the list of currently-defined matrices |
| _Eigen_Show_MatrixSpecs | Display the list of matrix Specs variables |
| _Eigen_Show_MatrixSpecs_Current | Display the current contents of MatrixSpecs |
| _Eigen_Show_ScalarOperators | Display the list of Cwise scalar operators |
| _Eigen_Show_UnaryOperators | Display the list of Cwise unary operators |
| _Eigen_Show_Warnings | Disable Warning-Proceed? dialogs in unexpected situations |
| _Eigen_Skewness | Return the global matrix skewness (asymmetry) |
| _Eigen_Skewness_Col | Return the skewness (asymmetry) for a specified matrix column |
| _Eigen_Skewness_Row | Return the skewness (asymmetry) for a specified matrix row |
| _Eigen_SplitMatrix_FromAcol | Split matrix A at a specified column into new matrices B and C |
| _Eigen_SplitMatrix_FromAcol_Transposed | Split matrix A at a specified column into new, transposed matrices B and C |
| _Eigen_SplitMatrix_FromArow | Split matrix A at a specified row into new matrices B and C |
| _Eigen_SplitMatrix_FromArow_Transposed | Split matrix A at a specified row into new, transposed matrices B and C |
| _Eigen_SSR | Compute the sum of *squared* residuals after obtaining a model fit |
| _Eigen_SSR_Col | Compute the sum of *squared* residuals after obtaining a model fit, for a specified column of observations matrix B |
| _Eigen_SSR_Colwise | Compute the sum of *squared* residuals after obtaining a model fit, per column of observations matrix B |
| _Eigen_Startup | Initialise an **Eigen4AutoIt** work environment |
| _Eigen_StDev | Return the global matrix standard deviation |
| _Eigen_StDev_Col | Return the standard deviation for a specified matrix column |
| _Eigen_StDev_Colwise | Return, in a Rowvector, the standard deviation for each column |
| _Eigen_StDev_Row | Return the standard deviation for a specified matrix row |
| _Eigen_StDev_Rowwise | Return, in a Colvector, the standard deviation for each row |
| _Eigen_Swap_Ablock_Ablock | Exchange the contents of two specified blocks of a single matrix |
| _Eigen_Swap_Ablock_Bblock | Exchange the contents of a specified block in one matrix with an equal-sized block in another matrix |
| _Eigen_Swap_Acol_Acol | Exchange the contents of two specified columns of a single matrix |

| | |
|---|---|
| _Eigen_Swap_Acol_Adiag | Exchange the contents of a specified column with the contents of the diagonal |
| _Eigen_Swap_Acol_Arow | Exchange the contents of a specified column and row of a single, square matrix |
| _Eigen_Swap_Acol_Bcol | Exchange the contents of a specified column in one matrix with a specified, equal-sized column in another matrix |
| _Eigen_Swap_Acol_Bdiag | Exchange the contents of a specified column with those of another matrix's diagonal |
| _Eigen_Swap_Acol_Brow | Exchange the contents of a specified column in one matrix with a specified, equal-sized row in another matrix |
| _Eigen_Swap_Adiag_Bdiag | Exchange the contents of the diagonals between two matrices |
| _Eigen_Swap_ALower_BLower | Exchange the contents of the lower triangular parts between two matrices |
| _Eigen_Swap_Areal_Aimag | Exchange the real and imaginary parts of a single complex matrix |
| _Eigen_Swap_Areal_Bimag | Exchange the real part of one complex matrix with imaginary part of another complex matrix |
| _Eigen_Swap_Arow_Adiag | Exchange the contents of a specified row with the contents of the diagonal |
| _Eigen_Swap_Arow_Arow | Exchange the contents of two specified rows of a single matrix |
| _Eigen_Swap_Arow_Bdiag | Exchange the contents of a specified column with the contents of another matrix's diagonal |
| _Eigen_Swap_Arow_Brow | Exchange the contents of a specified row in one matrix with a specified, equal-sized row in another matrix |
| _Eigen_Swap_AStrictlyLower_BStrictlyLower | Exchange the contents of the strictly-lower triangular part of one matrix with the equal-sized, strictly-lower triangular part of another matrix |
| _Eigen_Swap_AStrictlyUpper_BStrictlyLower | Exchange the contents of the strictly-upper triangular part of one matrix with the equal-sized, strictly-lower triangular part of another matrix |
| _Eigen_Swap_AStrictlyUpper_BStrictlyUpper | Exchange the contents of the strictly-upper triangular part of one matrix with the equal-sized, strictly-upper triangular part of another matrix |
| _Eigen_Swap_AUpper_ALower | Exchange the contents of the upper and lower triangular parts of a single, square matrix |
| _Eigen_Swap_AUpper_BLower | Exchange the contents of the upper triangular part of one matrix with the equal-sized, lower triangular part of another matrix |
| _Eigen_Swap_AUpper_BUpper | Exchange the contents of the upper triangular parts between two matrices |
| _Eigen_Test_EigenAssert | Trigger an Eigen assert failure message in Debug Mode |
| _Eigen_Transpose | Mirror-exchange all cell contents across the diagonal (flip dimensions) |
| _Eigen_Transpose_InPlace | Mirror-exchange all cell contents across the diagonal (flip dimensions); stored in-place |
| _Eigen_WriteMatrixValue | Write value to a specific matrix cell |

**Function Notes**

# Function Notes

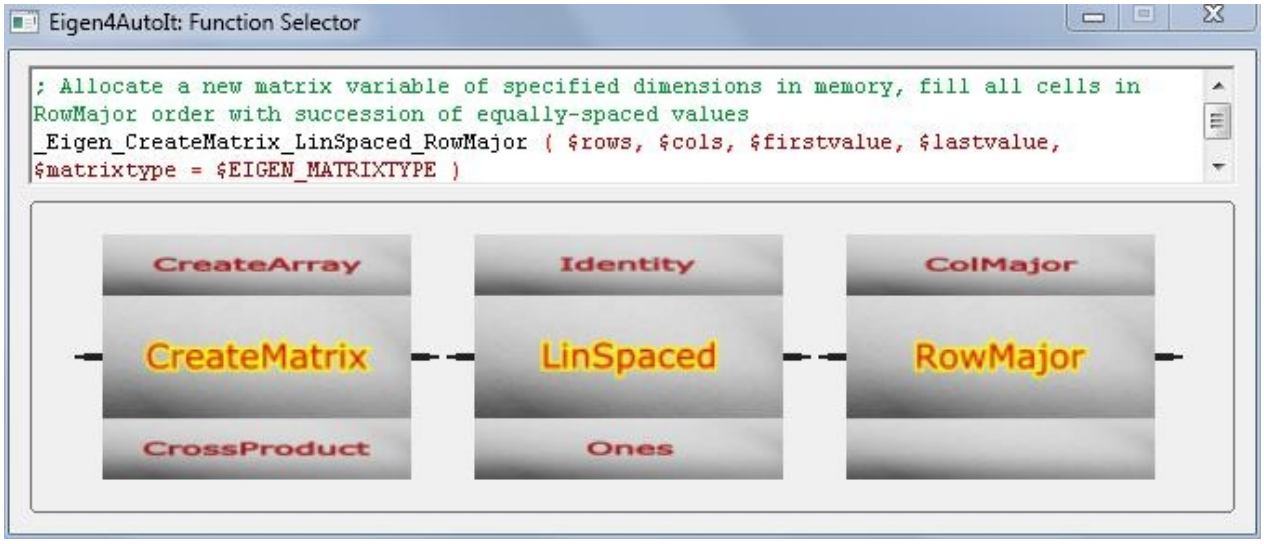
Most of **Eigen4AutoIt**'s functions follow an intuitive naming convention, consisting of up to four parts, separated by underscores. The first two parts are always present, whereas the third and fourth can each be absent, depending on the function's context. “Actions” are often a single verb or noun; notable exceptions are those that start with "Get," "Set," "Reset," "Release," and "Convert," as well as those whose next word is "Matrix" or "Array." In most cases, alternate phrasing with underscores in different places will be converted into the official name through a multitude of alias function wrappers.

If no target is specified, a function will act on the entire matrix (or matrices) supplied. Functions with *Diag* suffix act on the main diagonal only, *Col/Row* on a specified column or row, and *Block* on a rectangular area defined by startRow, startCol (both base-0), blockRows (height), and blockCols (width). *Colwise* and *Rowwise* act on the entire matrix, but on each col/row individually, storing the result in a separate vector. Suffix *_inPlace* signals that the result of the action is to be stored in the input matrix itself, replacing the original content. This will only work if that result fits (see Multiplication). *ColMajor* and *RowMajor* determine the order of *CwiseLinSpaced*; *Single* applies only to *MatrixSpecs*. Various other suffices are used only in the context of specifc functions.

| Prefix | Action | Suffix1 | Suffix2 |
|---|---|---|---|
| Eigen | Set* | Diag | inPlace |
| | Copy | Col | ColMajor |
| | MatrixSpecs | Row | RowMajor |
| | Cwise*Op | Block | Single |
| | Multiply | Colwise | |
| | ... | ... | ... |

A special tool called the ***Function Selector*** can quickly familiarise beginners with how function names are constructed. It displays three animated barrels that can be spun in either direction at the click of a mouse (above or below the barrel's front face; click multiple times for additional speed). The left-most barrel cycles through all Actions; as soon as it is stopped (by clicking the front face, or after "braking" by clicking on the side opposite from the current rotation direction), the other two barrels have their faces updated to reflect the various suffix members that this Action may have. In the panel above the barrels, any valid combination of an Action and 0-2 suffices results in the display of a short description, and the formal function expression with all parameters and default values. Any such function can then be copied to the Clipboard by pressing Ctrl-C (without having to select the displayed text first); whether the listed parameters are also copied depends on the parameter-mode, which is controlled by repeatedly pressing Ctrl-P to cycle thorugh the three options (no parameters, essential parameters only, all parameters). As a

bonus, This tool can also be used to add **Eigen4AutoIt** function tooltips and auto-completion to the **Scite** editor, by pressing Ctrl-A (whereas Ctrl-R removes these again from your Scite user profile).




**Eigen4AutoIt** functions, like **AutoIt** functions, are first class objects, meaning you can assign a function to a variable, and pass it around as an argument or return from another function. For example, instead of writing these two lines:

$matC = **_Eigen_Multiply_AB (** $matA, $matB **)**
**_Eigen_MatrixSpecs (** $matC **)**

one could nest the second call into the first, obviating the need to store the matrix ID of the intermediate result in a separate variable, like this:

**_Eigen_MatrixSpecs ( _Eigen_Multiply_AB (** $matA, $matB **) )**

But even though variable $matC is no longer used, a separate entry in the MatrixList is still created for each intermediate results matrix. You can maintain full control of your memory resources by placing the ListMarker just before starting some multi-step procedure with several intermediates, and releasing all matrices created afterwards, with the exception of the final result(s) you wish to keep. See **_Eigen_ReleaseFromMarker_Except()** for details.

Many functions contain optional parameters that can be omitted. If you wish to specify an optional parameter, however, **all preceding parameters must also be specified!** This is a standard **AutoIt** convention.

For example, consider **_Eigen_SetActiveMatrix(** $matrixstring**[,** $resetAll**[,** $releaseExisting **]] )**. If you wish to specify the last boolean flag *$releaseExisting*, you **must** specify $resetAll too. If you do not know what to parse, you can always use the Default keyword, to use its default setting.

Almost all **Eigen4AutoIt** functions indicate success / failure as a boolean **return value** (True / False), while also setting the **@error flag**. All error codes are listed in the Appendix. Matrix decompositions and related functions additionally return a status code upon failure , which is incorporated into **Eigen4AutoIt**'s own error message. For a listing, see the table in the Appendix.

Some guidelines:

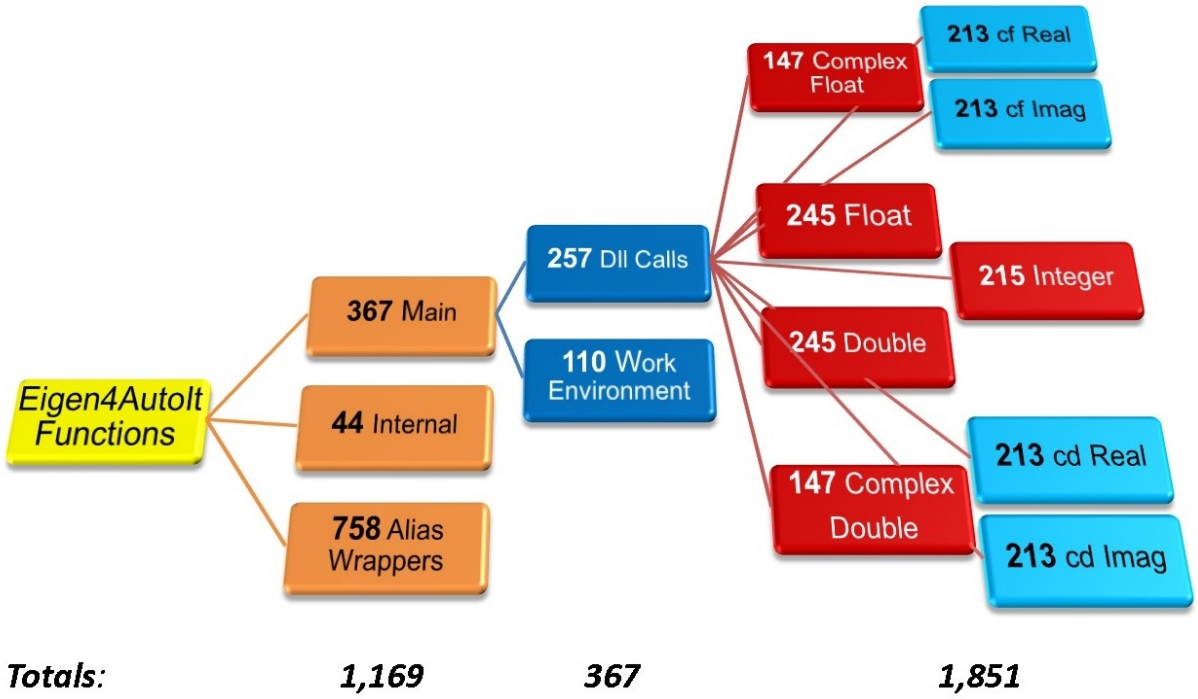
@error is always set to 0 when entering a function.

@error = 0 is almost always success
Return = varies (often it is the target matrix ID), but is typically non-zero for success.

```
If someFunc() Then  ;...function worked
If Not someFunc() Then  ;...function failed

$matA = _Eigen_LoadMatrix ( "C:\someMatrixFile.bin" )
If @error = -1 Then ; matrix was not properly loaded
```

**If a function sets the @error flag, you should ideally check it before using a return value - if @error indicates an error then the function return value may be undefined or invalid.**

Currently, the **Eigen4AutoIt** environment comprises **367** main functions, consisting of **257** dll-calling wrappers and **110** work environment functions (such as _Eigen_CreateMatrix() ). The function wrappers themselves call one of nine type-specific function **variants** (NB not all variants are available for each type), which total **1,851** altogether. Each of these variants exists in two versions, for Debugging mode (with internal error checking) and for Computing Mode (without such checks). Note that separate variants may exist that act on a complex matrix as entity, its real part, or its imaginary part (that is, the rightmost cyan subsections do *not* overlap with the red subsections). Lastly, to maximise ease-of-use, an additional **758** alias wrappers are provided that allow the same main function to be accessed using variations of the formal function name.






## Complex Function Notes

# Complex Function Notes

Most of **Eigen4AutoIt**'s functions follow an intuitive naming convention, consisting of up to four parts, separated by underscores. Complex functions constitute a superset of these, with an optional **fifth** suffix: **_Real** or **_Imag**. Due to limitations of **Eigen** itself, usage is not as straightforward as this may sound. When the environment's matrix type is complex (either "complexfloat" or "complexdouble"), the following four categories of functions are distinguished:

- **not** applicable at all
  - relates to work environment, part of matrix management, some decompositions, and all statistics
  - Examples: **Convert*, Show*, Decomp_Choleski, StDev***
- **fully** applicable
  - function accepts **both** complex matrices ***and*** their real or imaginary part(s) separately, to work on
  - Examples: **Copy*, Multiply*, CreateMatrix_From***
- **no** complex ***parts*** supported
  - function accepts **only** complex matrices, ***not*** their real or imaginary part(s) separately, to work on
  - Examples: **Copy_A_ToBimag, Copy_A_ToBreal, Swap_Real_Imag, Decomp***
- ***only*** complex ***parts*** supported
  - function accepts **only** real or imaginary part of complex matrices to work on separately, **not** the full complex matrix as a single entity
  - **Condit*, Cwise*, DotProduct, MatrixSpecs***

Thus not all functions accept complex inputs, and of those that do, some accept only complex matrices as single entity, others act only on either the real or imaginary part(s) separately, and some can do both. For an overview of which **variants** are supported for any matrixtype-specific dll function, call _Eigen_Show_DllFunctions(). Each column in the displayed array represents a function variant, with suffix **f** = float, **d** = double, **i** = integer, **c** = complex, **c*r** = real part, and **c*i** = imaginary part.
For example, if a function is marked "True" in the columns for suffices "_f" and "_d" only, then it supports no complex inputs at all; if it has True" in the columns for suffices "_cfr" and "_cfi", then you can call wrapper functions **<function_name>_Real** and **<function_name>_Imag** to act only on the real or imaginary part respectively; if it has True" only in the columns for suffices "_cf" and "_cd" then you can call **<function_name>** directly while parsing ID's of complex matrices, but acting separately on the real or imaginary part (by appending **_Real** or **_Imag**) is not supported.

In summary, after setting your work environment to a complex matrix type, and then parsing complex matrices:

- call **<function>** itself to act on the entire complex matrix (if supported)
- call **<function>_Real** to act only on the real part of a complex matrix (if supported)
- call **<function>_Imag** to act only on the imaginary part of a complex matrix (if supported)

General overview of supported function variants per section (with exceptions!):

| Function | real float / double | integer | complex float / double | complex parts |
|---|---|---|---|---|
| Create / Convert | yes | yes | yes | NO |
| Split/Set/Copy/Swap | yes | yes | yes | yes |
| SetLinSpaced | yes | yes | NO | yes |
| Cwise/Condit Ops | yes | yes | NO | yes |

| | | | | |
|---|---|---|---|---|
| **MatrixSpecs** | yes | yes | NO | yes |
| **Transpose/Reverse** | yes | yes | yes | yes |
| **Inverse** | yes | NO | yes | yes |
| **Multiply** | yes | yes | yes | yes |
| **Decompositions** | yes | NO | *some* | NO |
| **Eigensolvers** | *some* | NO | yes | NO |
| **Statistics** | yes | NO | NO | NO |

Other peculiarities to keep in mind:

- **MatrixSpecs_Colwise_Single** and **MatrixSpecs_Rowwise_Single** are not supported for *any* complex variant.
- **_MatrixDisplay** will by default display the **real** part of a complex matrix only; to inspect the imaginary part, call **_MatrixDisplay_Imag** instead
- Ditto for **CopyArrayData_ToMatrix** and **CopyMatrixData_ToArray**; these act on the real part by default; use the **_Imag** suffix to access the imaginary part instead
- Decompositions that explicitly act on real matrices are obviously not supported for complex types; this concerns: **Decomp_Choleski, Decomp_RealSchur, Decomp_RealQZ**.
- Some decomposition outputs (e.g., **JacobiSVD** eigenvalues, returned in generic matrix **S**) are always of type real, even if the input is complex. In a complex work environment, the data are then stored in the complex container's real part.
- **CreateMatrix** and its prefill cousins (**CreateMatrix_Ones**, **CreateMatrix_Random**, etcetera) can act type-specifically, irrespective of the current work environment. So you can create a real float matrix in a complex float environment, for example,or a complex float matrix in a real float environment, by specifying the optional third parameter (the type-string, e.g., "float").
- All type-independent file I/O functions (**LoadMatrix**, **SaveMatrix**, **ConvertMatrixFile***), as well as the external **MatrixFileConverter** utility, can process complex matrices; the associated files require about twice as much storage space as their real equivalents do at the same level of precision. Note that it is not possible to apply these I/O functions separately on the real or imaginary part of a complex matrix. It is also not possible to store just the real part or imaginary part of a complex matrix directly. You can, however, copy their contents to (or from) a real matrix that can be saved and loaded separately.
- Conversions from complex to real will discard the imaginary part of a complex matrix, losing information.
- Conversions from real to complex will fill the imaginary part of a complex matrix with zeroes.
- **Eigen** stores complex data interleaved, that is, the real and imaginary parts are stored per matrix cell. Some other computational software packages (e.g., MatLab) store complex data split, that is, first all real parts, then all imaginary parts of the matrix cells. Neither method is inherently better; each has their own advantages and disadvantages. Internal functions **_ConvertMatrix_ToSplit** and **_ConvertMatrix_ToInterleaved** can be accessed though the **MatrixFileConverter** to facilitate importing and exporting such data sets.



### Work Environment

# Work Environment

You'll need to consider **two** settings that define an **Eigen4AutoIt** work environment:

- **Matrix Type** (afftect single versus double precision, and real versus complex values)
  Matrices store numbers as binary data at some given level of precision. Currently, **Eigen4AutoIt** supports **real floats**, the default, **real doubles**, **integers**, **complex floats** and **complex doubles** for computation. Now, the First Rule of Eigen Club is: **DO NOT MIX MATRIX TYPES IN EIGEN CALLS!** Eigen cannot handle automatic type conversion, so choose a precision level for a session and stick with it. You can do this by calling either: **_Eigen_SetMatrixType( "float" )** c.q. **_Eigen_SetMatrixType( "double" )** which itself calls **_Eigen_Startup()** if the parsed type differs from the curernt one, or by calling **_Eigen_Startup( "double" )** directly.

| Type ID | Matrix Type | Description |
|---|---|---|
| 0 | "realfloat" | 4 bytes, 6 significant digits; default precision: ***1.0 e-6*** |
| 1 | "integer" | 4 bytes; fixed precision: ***1.0 e0*** |
| 2 | "realdouble" | 8 bytes, 12 significant digits; default precision: ***1.0 e-12*** |
| 4 | "complexfloat" | 2 x 4 bytes; stored pairwise; default precision: ***1.0 e-6*** |
| 6 | "complexdouble" | 2 x 8 bytes; stored pairwise; default precision: ***1.0 e-12*** |

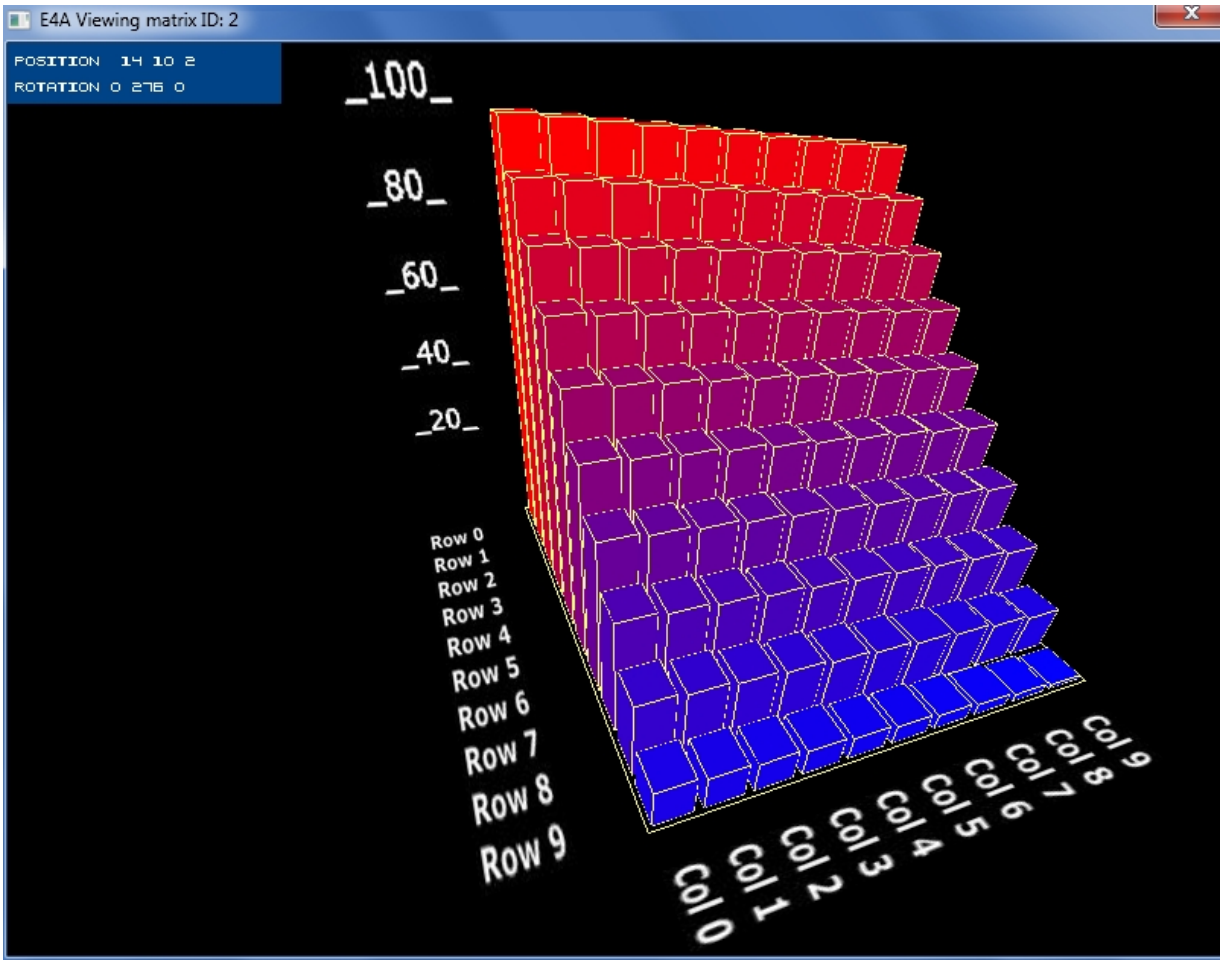
- **Debug** mode versus **Computing** mode
  There are two versions of Eigen's dense matrix module: ***EigenDense_Debug.dll*** and ***EigenDense.dll***; the first contains extensive error/bounds checks and asserts; the second just blazes forth and will fail and/or return invalid results if *anything* goes wrong. Please **always** run new scripts in Debug mode first (the default). Note that assert failures generate a error message on screen that requires user interaction. Secondly, Debug mode slows down processing a *lot*; please do not perform benchmarking tests in this mode.
  Once you're satisfied that your script works as expected, you can switch to Computing Mode with: **_Eigen_DebugMode_Off()**.

The work environment furthermore comprises a number of internal global variables and constants, listed in the Appendix. Most of these are set through higher-level functions; you should not need to engage with them directly.

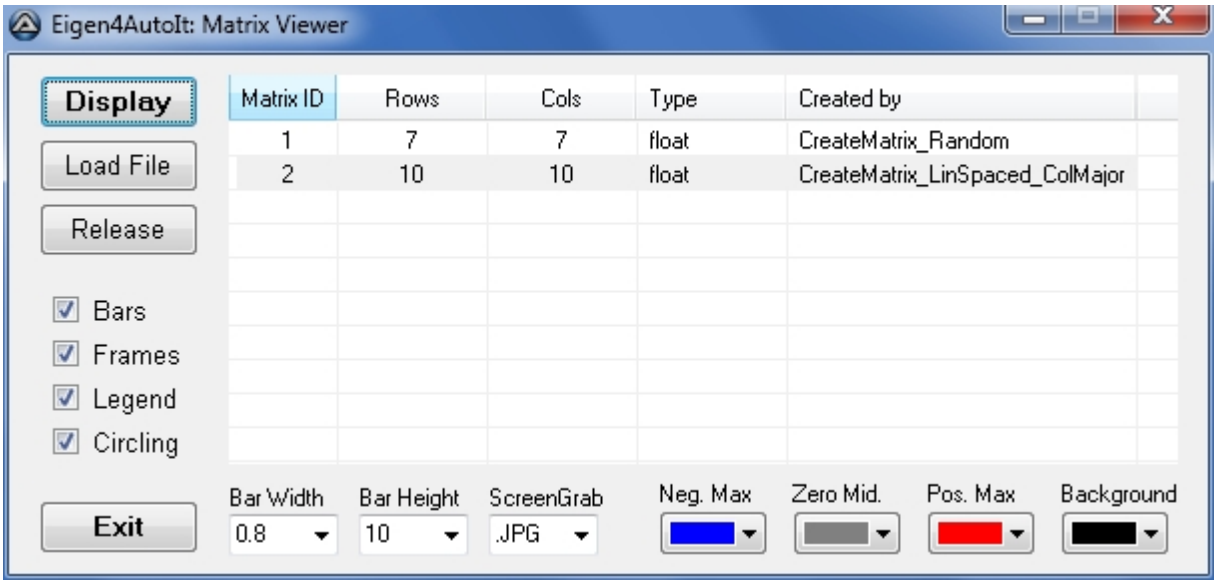
If you wish to explore **Eigen4AutoIt**'s multiprocessing capabilities, you'll need to download and install the Pool environment from the **AutoIt** Example Scripts forum. See the multi-processing tutorial and Pool's own documentation for further details.

Several stand-alone **E4A** utilities offer additional features for user convenience:

- the **MatrixFileConverter** tool is described here
- the 3D animated **FunctionSelector** tool is described here
- the 3D animated **MatrixViewer** tool is described below.




The **MatrixViewer** is a simple graphical user interface (GUI) that allows you to visualise a 2D matrix as a 3D bar graph. Cell values are represented in the vertical dimension and/or through the colour scale (with separate user-defined colour gradients for positive and negative values. Using mouse, cursor (arrow) keys, and letter keys (W/S to zoom, A/D to strafe, and Q/E to control viewpoint elevation) you can then move around this object in real-time, as if you are flying around it. Alternatively you can press C (or Shift-C) to start/stop circling around the matrix object in (anti-)clockwise direction. At any point, you can press G to save the image in four different formats for further processing. Press <Home> to reset the camera to its original vantage point, and <Esc> to terminate the separate viewport window, and return to the GUI, from which various other graph settings can be controlled, such as the width and maximum height of the bars. Press <Ctrl-D> to reload the default settings; your own preferred settings are saved in local file **MatrixViewer.ini**. When the GUI is incorporated in your own **E4A** environment (using MVinclude.au3), it can also be used to inspect the current contents of the MatrixList, to load matrix files into it, and to release existing matrices from memory.






## CleanUp

Function Reference

# _Eigen_CleanUp

Cleans up an Eigen4AutoIt work environment initialised with _Eigen_StartUp()

```
#include <Eigen4AutoIt.au3>
_Eigen_CleanUp ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Closes any open EigenDense dll handle, and clears the entire $MatrixList, releasing all currently defined matrices.

## Related

*StartUp*(), *DebugMode_Off*(), *DebugMode_On*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_EnvironmentVars()

_Eigen_CleanUp()
```



### Debugmode_Off

Function Reference

# _Eigen_DebugMode_Off

Switch from Debug mode to Computing mode

```
#include <Eigen4AutoIt.au3>
_Eigen_DebugMode_Off ( [$releaseAll = True ] )
```

## Parameters

| | |
|---|---|
| $releaseAll | **[optional]** **True**: release all currently defined matrices from memory; **False**: leave any existing matrices untouched |

## Return Value

Success: True

Failure: n/a

## Remarks

Switches to Computing Mode and calls **_Eigen_Startup()** again, meaning all dll calls henceforth use ***EigenDense.dll***.
Computing mode implies that no checks are performed by **Eigen** itself (which speeds up processing), but no errors will be caught or flagged by the dll.

By default, any existing matrices will be released from memory upon restarting the environment; to retain these, set boolean $releaseAll to False.

## Related

*StartUp*(), *CleanUp*(), *DebugMode_On*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_SetDebugmode_On()
_Eigen_Show_EnvironmentVars()

_Eigen_SetDebugmode_Off()
_Eigen_Show_EnvironmentVars()

_Eigen_CleanUp()
```





# _Eigen_DebugMode_On

Switch from Computing mode to Debug mode

```
#include <Eigen4AutoIt.au3>
_Eigen_DebugMode_On ( [$releaseAll = True ] )
```

## Parameters

| | |
|---|---|
| $releaseAll | **[optional]** **True**: release all currently defined matrices from memory; **False**: leave any existing matrices untouched |

## Return Value

Success: True

Failure: n/a

## Remarks

Switches to Debug Mode and calls **_Eigen_Startup()** again, meaning all dll calls henceforth use ***EigenDense_Debug.dll***.
Debug mode implies that numerous extra checks are performed by Eigen itself (which does slow down processing), and if any error is caught, the dll produces a special assert failure error message before returning control to **Eigen4AutoIt** (Which will then produce its own error message that the call has failed).

By default, any existing matrices will be released from memory upon restarting the environment; to retain these, set boolean $releaseAll to False.

## Related

*StartUp*(), *CleanUp*(), *DebugMode_Off*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_SetDebugmode_On()
_Eigen_Show_EnvironmentVars()

_Eigen_SetDebugmode_Off()
_Eigen_Show_EnvironmentVars()

_Eigen_CleanUp()
```



### Hide_Errors

Function Reference

# _Eigen_Hide_Errors

Disable all non-fatal error messages

```
#include <Eigen4AutoIt.au3>
_Eigen_Hide_Errors ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Use this function in combination with _Eigen_Hide_Warnings() to ensure interruption-free processing, regardless of whether individual operations fail (not recommended). This is useful for high- throughput computing tasks where the failure of individual jobs is insignificant. and bulk processing should continue regardless.

Fatal errors are always displayed.

## Related

*Show_Errors*(), *Hide_Warnings*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_SetMatrixType ( "float" )          ; calls _Eigen_Startup()
$matA = _Eigen_CreateMatrix ( 3, 4 )      ; initial type = float

_Eigen_Show_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 4, 3, "int" )    ; warning triggered

_Eigen_Hide_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 2, 6, "double" ); NO warning triggered

_Eigen_Cleanup()
```





# _Eigen_Hide_Warnings

Disable Warning-Proceed? dialogs in unexpected situations

```
#include <Eigen4AutoIt.au3>
_Eigen_Hide_Warnings ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Most warnings involve the handling of matrix variable types that differ from **Eigen4AutoIt**'s current work

environment's matrix type.

## Related

*Show_Warnings()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_SetMatrixType ( "float" )          ; calls _Eigen_Startup()
$matA = _Eigen_CreateMatrix ( 3, 4 )      ; initial type = float

_Eigen_Show_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 4, 3, "int" )    ; warning triggered

_Eigen_Hide_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 2, 6, "double" ); NO warning triggered

_Eigen_Cleanup()
```





# _Eigen_IsComplex

Return True if the type of a parsed matrix ID is complex float or complex double

```
#include <Eigen4AutoIt.au3>
_Eigen_IsComplex ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: True / False, reflecting bit 2 of the matrix variable type

Failure: -1, and sets the @error flag to non-zero.

## Remarks

None.

## Related

None.

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
$matB = _Eigen_CreateMatrix ( 3, 4, "complexfloat" )
_Eigen_Show_MatrixList()

$typeA = _Eigen_GetMatrixType ( $matA )
$typeB = _Eigen_GetMatrixType ( $matB )

MsgBox ( 0, "matrix A", "type: " & $typeA  "; IsComplex? " &
_Eigen_IsComplex( $matA ) )
MsgBox ( 0, "matrix B", "type: " & $typeB  "; IsComplex? " &
_Eigen_IsComplex( $matB ) )

_Eigen_CleanUp()
```



### ReleaseFromIndex

Function Reference

# _Eigen_ReleaseFromIndex

Release the indexed matrix from memory, as well as all matrices created thereafter

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseFromIndex ( $index )
```

## Parameters

| | |
|---|---|
| $index | row index to global array $MatrixList, in which all matrices are stored |

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Best practice would suggest using a ListMarker for this task, but some scenarios may call for additional control.
Note that unlike the Listmarker (which identifies the *last* matrix to *preserve*), $index identifies the *first* matrix to *release*.

## Related

*SetListMarker*(), *ResetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex*(), *ReleaseFromIndex_ToIndex_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )
$matD = _Eigen_CreateMatrix ( 4, 4 )
$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseFromIndex ( 4 ) ; any matrix created from index 4 (inclusive) is
released
_Eigen_Show_MatrixList()

_Eigen_CleanUp()              ; all remaining matrices are now also released
```



### ReleaseFromIndex_Except

Function Reference

# _Eigen_ReleaseFromIndex_Except

Release the indexed matrix from memory, as well as all matrices created thereafter, with exceptions

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseFromIndex_Except ( $index[, $mat1 = 0[, $mat2 = 0[, $mat3 =
0[, $mat4 = 0[, $mat5 = 0[, $mat6 = 0[, $mat7 = 0[, $mat8 = 0[, $mat9 =
0[, $mat10 = 0 ]]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $index | row index to global array $MatrixList, in which all matrices are stored |
| $mat# | **[optional]** matrix ID of the matrix to **preserve** |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

Each matrix ID equal or larger than index will be released from memory, **with the exception of the parsed matrix IDs**.

Best practice would suggest using a ListMarker for this task, but some scenarios may call for additional control.
Note that unlike the Listmarker (which identifies the *last* matrix to *preserve*), $index identifies the *first* matrix to *release*.

## Related

*SetListMarker*(), *ResetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex*(), *ReleaseFromIndex_ToIndex*(), *ReleaseFromIndex_ToIndex_Except*(), *ReleaseMatrix_All_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )
$matD = _Eigen_CreateMatrix ( 4, 4 )
$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()

; any matrix created from index 4 is released, except specified matrices E
and H
_Eigen_ReleaseFromIndex_Except( 4, $matE, $matH )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()  ; all remaining matrices are now also released
```



ReleaseFromIndex_ToIndex

# _Eigen_ReleaseFromIndex_ToIndex

Release from memory all matrices within the specified index range

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseFromIndex_ToIndex ( $index_first, $index_last )
```

## Parameters

| | |
|---|---|
| $index_first | start of the range of matrices to release in global array $MatrixList |
| $index_last | end of the range of matrices to release in global array $MatrixList |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

Each matrix ID between $index_first and $index_last (inclusive) will be released from memory.

Best practice would suggest using a ListMarker for this task, but some scenarios may call for additional control.
Note that unlike the Listmarker (which identifies the *last* matrix to *preserve*), $index_first identifies the *first* matrix to *release*.

## Related

*SetListMarker*(), *ResetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )
$matD = _Eigen_CreateMatrix ( 4, 4 )
$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()
```

```
; all matrices from index 4 to 7 inclusive are released
_Eigen_ReleaseFromIndex_ToIndex( 4, 7 )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()  ; all remaining matrices are now also released
```





# _Eigen_ReleaseFromIndex_ToIndex_Except

Release from memory all matrices within the specified index range, with exceptions

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseFromIndex_toIndex_Except ( $index_first, $index_last[, $mat1 = 0[, $mat2 = 0[, $mat3 = 0[, $mat4 = 0[, $mat5 = 0[, $mat6 = 0[, $mat7 = 0[, $mat8 = 0[, $mat9 = 0[, $mat10 = 0 ]]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $index_first | start of the range of matrices to release in global array $MatrixList |
| $index_last | end of the range of matrices to release in global array $MatrixList |
| $mat# | **[optional]** matrix ID of the matrix to **preserve** |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

Each matrix ID between $index_first and $index_last (inclusive) will be released from memory, **with the exception of the parsed matrix IDs**.

Best practice would suggest using a ListMarker for this task, but some scenarios may call for additional control.
Note that unlike the Listmarker (which identifies the *last* matrix to *preserve*), $index_first identifies the *first* matrix to *release*.

## Related

*SetListMarker*(), *ResetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex*(), *ReleaseMatrix_All_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )
$matD = _Eigen_CreateMatrix ( 4, 4 )
$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()

; all matrices from index 4 to 7 inclusive are released, except specified
matrix E
_Eigen_ReleaseFromIndex_ToIndex_Except( 4, 7, $matE )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()  ; all remaining matrices are now also released
```



### ReleaseFromMarker

Function Reference

# _Eigen_ReleaseFromMarker

Release from memory all matrices created after the ListMarker was set

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseFromMarker ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Setting the ListMarker with **_Eigen_SetListMarker()** stores the current-last index of global array $MatrixList

(in which all matrices are stored) in global variable $MatrixListMarker. Calling **_Eigen_ReleaseFromMarker()** afterwards will then release all matrices created subsequently to this point, and clears the ListMarker again.

## Related

*SetListMarker*(), *ResetListMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex*(), *ReleaseFromIndex_ToIndex_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )

_Eigen_SetListMarker()  ; ListMarker is now set to index 3

$matD = _Eigen_CreateMatrix ( 4, 4 )
$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseFromMarker()    ; any matrix created after the ListMarker is
released
_Eigen_Show_MatrixList()

_Eigen_CleanUp()        ; all remaining matrices are now also released
```



### ReleaseFromMarker_Except

Function Reference

# _Eigen_ReleaseFromMarker_Except

Release all matrices from memory beyond the ListMarker, with exceptions

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseFromMarker_Except ( [$mat1 = 0[, $mat2 = 0[, $mat3 = 0[,
$mat4 = 0[, $mat5 = 0[, $mat6 = 0[, $mat7 = 0[, $mat8 = 0[, $mat9 = 0[,
$mat10 = 0 ]]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $mat# | **[optional]** matrix ID of the matrix to **preserve** |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

Setting the ListMarker with **_Eigen_SetListMarker()** stores the current-last index of global array $MatrixList (in which all matrices are stored) in another global variable called $MatrixListMarker. Calling **_Eigen_ReleaseFromMarker_Except()** afterwards will release all matrices created subsequently to this point, **with the exception of the parsed matrix IDs**.

## Related

*SetListMarker*(), *ResetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromIndex*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex*(), *ReleaseFromIndex_ToIndex_Except*(), *ReleaseMatrix_All_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )

_Eigen_SetListMarker()  ; ListMarker is now set to index 3

$matD = _Eigen_CreateMatrix ( 4, 4 )
$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()

; any matrix created after the ListMarker is released, except specified
matrices E and H
_Eigen_ReleaseFromMarker_Except( $matE, $matH )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()        ; all remaining matrices are now also released
```



## ResetListMarker

Function Reference

# _Eigen_ResetListMarker

Clear the ListMarker

```
#include <Eigen4AutoIt.au3>
_Eigen_ResetListMarker ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Setting the ListMarker stores the current-last index of global array $MatrixList in global variable $MatrixListMarker, to enable the eventual release of all matrices created subsequently to this point. Calling _Eigen_ResetListMarker clears the marker, and fills $MatrixListMarker with **-1** ("missing value"). No matrices are affected in any way.

## Related

*SetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex*(), *ReleaseFromIndex_ToIndex_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )

_Eigen_SetListMarker()        ; ListMarker is now set to index 3

$matD = _Eigen_CreateMatrix ( 4, 4 )

_Eigen_ResetListMarker()      ; ListMarker is now undefined again (-1)

$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )

_Eigen_SetListMarker()        ; ListMarker is now set to index 6
```

```
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseFromMarker()    ; any matrix created after the ListMarker is
released
_Eigen_Show_MatrixList()

_Eigen_CleanUp()              ; all remaining matrices are now also released
```





# _Eigen_SetListMarker

Store the current-last index of the MatrixList for future reference

```
#include <Eigen4AutoIt.au3>
_Eigen_SetListMarker ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Global array $MatrixList is the central repository of all matrix variables (structs), together with their dimensions, type, and memory address pointer. Whenever a new matrix is created, a new entry is added to this list, and its index constitutes the matrix ID that most functions return upon success.

Setting the ListMarker stores the current-last index in global variable $MatrixListMarker, to enable the eventual release of all matrices created subsequently to this point.

## Related

*ResetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex*(), *ReleaseFromIndex_ToIndex_Except*()

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 1 )
$matB = _Eigen_CreateMatrix ( 2, 2 )
$matC = _Eigen_CreateMatrix ( 3, 3 )

_Eigen_SetListMarker()        ; ListMarker is now set to index 3

$matD = _Eigen_CreateMatrix ( 4, 4 )
$matE = _Eigen_CreateMatrix ( 5, 5 )
$matF = _Eigen_CreateMatrix ( 6, 6 )
$matG = _Eigen_CreateMatrix ( 7, 7 )
$matH = _Eigen_CreateMatrix ( 8, 8 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseFromMarker()    ; any matrix created after the ListMarker is
released
_Eigen_Show_MatrixList()

_Eigen_CleanUp()              ; all remaining matrices are now also released
```





# _Eigen_SetMatrixType

Select a matrix variable type prior to a calling _Eigen_Startup()

```
#include <Eigen4AutoIt.au3>
_Eigen_SetMatrixType ( $matrixType = $EIGEN_MATRIXTYPE )
```

## Parameters

| | |
|---|---|
| $matrixType | **[optional]** "float" (default), "double", "int", "complexfloat", or "complexdouble" |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

This function performs different tasks depending on what is parsed:

If no parameter is parsed, or the parsed matrix type matches the current environment matrix type, then all type-dependent global variables are simply reset to their defaults, including rounding control (**$EIGEN_DECIMALS**).

If the parsed (valid) matrix type does **not** match the current environment type, this functions instead calls _Eigen_StartUp ( $matrixType, False ), which changes the environment's matrix type while retaining all existing matrices (which if of different type from the environment's new matrix type, will have to be individually type-converted before they can partake in Eigen calls in the changed environment). To change the matrix type of any specific matrix (in memory or on file) independently of the current work environment settings, see section "File I/O and Type Conversion". If, instead, you wish to make a clean break (releasing all previously-created matrices), simply call _Eigen_StartUp ( $matrixType ) instead of _Eigen_SetMatrixType( $matrixType ).

## Related

*Startup(), Cleanup(), DebugMode_Off(), DebugMode_On()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_SetMatrixType ( "double" )   ; calls _Eigen_Startup()
_Eigen_Show_EnvironmentVars()

_Eigen_SetMatrixType ( "float" )
_Eigen_Show_EnvironmentVars()

_Eigen_Cleanup()
```





# _Eigen_Show_AllFunctions

Display a list of Eigen4AutoIt's functions, with brief description

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_AllFunctions ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Lists all official function names, with a brief description.

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

## Related

None.

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_AllFunctions()

_Eigen_Cleanup()
```



### Show_Constants_Nist

Function Reference

# _Eigen_Show_Constants_Nist

Display a list of Eigen4AutoIt's physics constants, compiled by NIST

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_Constants_Nist ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

The listed global constants were compiled by the U.S. National Institute of Standards and Technology (last updated: 2010), and are available here.

To use these constants in your own scripts, add prefix **$E4AC_Nist_** to the variable names in the left-most

column of the table. For example, the full name of the first Nist constant is: $E4AC_Nist_SILICON_LATTICE_SPACING.

As both the Nist and Sykora2 compilations contain physics constants (mostly particle physics and astrophysics), there is considerable overlap between the two. Duplicate entries have not been pruned for practical reasons (future updates), and because terminology often differs, and users may prefer one over the other. Moreover, The Nist list is from 2010, whereas Sykora2's was last updated in 2012 (includes Higgs Boson data). The number of significant digits may also differ between these two lists.

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

## Related

*Show_Constants_Sykora1*(), *Show_Constants_Sykora2*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_Constants_Nist()

_Eigen_Cleanup()
```





# _Eigen_Show_Constants_Sykora1

Display a list of Eigen4AutoIt's mathematical constants, compiled by Sykora

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_Constants_Sykora1 ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

The listed global constants were compiled by Stanislav Sýkora, and are available here.

To use these constants in your own scripts, add prefix **$E4AC_Sykora1_** to the variable names in the left-most column of the table. For example, the full name of the first Sykora1 constant is: $E4AC_Sykora1_PI.

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

## Related

*Show_Constants_Sykora2*(), *Show_Constants_Nist*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_Constants_Sykora1()

_Eigen_Cleanup()
```



### Show_Constants_Sykora2

Function Reference

# _Eigen_Show_Constants_Sykora2

Display a list of Eigen4AutoIt's scientific constants, compiled by Sykora

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_Constants_Sykora2 ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

The listed global constants were compiled by Stanislav Sýkora, and are available here.

To use these constants in your own scripts, add prefix **$E4AC_Sykora2_** to the variable names in the left-most column of the table. For example, the full name of the first Sykora2 constant is:

$E4AC_Sykora2_LIGHTSPEED.

As both the Sykora2 and Nist compilations contain physics constants (mostly particle physics and astrophysics), there is considerable overlap between the two. Duplicate entries have not been pruned for practical reasons (future updates), and because terminology often differs, and users may prefer one over the other. Moreover, The Nist list is from 2010, whereas Sykora2's was last updated in 2012 (includes Higgs Boson data). The number of significant digits may also differ between these two lists.

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

## Related

*Show_Constants_Sykora1*(), *Show_Constants_Nist*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_Constants_Sykora2()

_Eigen_Cleanup()
```





# _Eigen_Show_DllFunctions

Display a list of Eigen4AutoIt's supported matrixtype-specific function variants

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_DllFunctions ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Lists all suffix-based variants of matrixtype-specific dll function call names; True = supported. The available

variants are:

| | |
|---|---|
| **_f** | float (single precision real) |
| **_d** | double (double precision real) |
| **_i** | integer |
| **_cf** | complex float (single precision complex) |
| **_cd** | complex double (double precision complex) |
| **_cfr** | real part of complex float |
| **_cfi** | imaginary part of complex float |
| **_cdr** | real part of complex double |
| **_cdi** | imaginary part of complex double |

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

## Related

None.

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_DllFunctions()

_Eigen_Cleanup()
```



### Show_EnvironmentVars

Function Reference

# _Eigen_Show_EnvironmentVars

Display a list of Eigen4AutoIt's current environment settings

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_EnvironmentVars ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |

Failure: n/a

## Remarks

The content of the listed global variables is determined by the way the **Eigen4AutoIt** environment is currently initialised.

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

## Related

*StartUp()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_SetMatrixType ( "double" )   ; calls _Eigen_Startup()
_Eigen_Show_EnvironmentVars()

_Eigen_SetMatrixType ( "float" )
_Eigen_Show_EnvironmentVars()

_Eigen_Cleanup()
```



### Show_Errors

Function Reference

# _Eigen_Show_Errors

Display all non-fatal error messages

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_Warnings ()
```

## Parameters

None.

## Return Value

Success: True

Failure: n/a

## Remarks

By default, all errors trigger an error message display. Use this function after calling _Eigen_Hide_Errors() to re-enable the display of messages for non-fatal errors.

Fatal errors are always displayed.

## Related

*Hide_Errors(), Show_Warnings()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_SetMatrixType ( "float" )          ; calls _Eigen_Startup()
$matA = _Eigen_CreateMatrix ( 3, 4 )      ; initial type = float

_Eigen_Show_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 4, 3, "int" )    ; warning triggered

_Eigen_Hide_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 2, 6, "double" ); NO warning triggered

_Eigen_Cleanup()
```





# _Eigen_Show_MatrixList

Display the MatrixList array, in which all defined matrices are stored

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_MatrixList ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Global array $MatrixList is the central repository of all matrix variables (structs), together with their dimensions, type, and memory address pointer. Whenever a new matrix is created, a new entry is added to this list, and its index constitutes the matrix ID that most functions return upon success. Whenever a matrix is released from memory, the associated array entry is blanked, but not removed. Controlled release of a range of matrices is achieved with the aid of a ListMarker (see the related functions listed below).

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

## Related

*SetListMarker*(), *ResetListMarker*(), *ReleaseFromMarker*(), *ReleaseFromMarker_Except*(), *ReleaseFromIndex*(), *ReleaseFromIndex_Except*(), *ReleaseFromIndex_ToIndex*(), *ReleaseFromIndex_ToIndex_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
_Eigen_Show_MatrixList()

$matA = _Eigen_CreateMatrix ( 1, 2 )
$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseMatrix ( $matA )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()
_Eigen_Show_MatrixList()
```





# _Eigen_Show_Warnings

Enable Warning-Proceed? dialogs in unexpected situations

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_Warnings ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Most warnings involve the handling of matrix variable types that differ from **Eigen4AutoIt**'s current work environment's matrix type.

## Related

*Hide_Warnings()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_SetMatrixType ( "float" )          ; calls _Eigen_Startup()
$matA = _Eigen_CreateMatrix ( 3, 4 )      ; initial type = float

_Eigen_Show_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 4, 3, "int" )    ; warning triggered

_Eigen_Hide_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 2, 6, "double" ); NO warning triggered

_Eigen_Cleanup()
```





# _Eigen_StartUp

Initialise an Eigen4AutoIt work environment

```
#include <Eigen4AutoIt.au3>
_Eigen_StartUp ( [$matrixType = $EIGEN_MATRIXTYPE[, $releaseAll =
```

```
True ]] )
```

## Parameters

| | |
|---|---|
| $matrixType | **[optional]** "float", "double" (single or double precision), "int", "complexfloat", "complexdouble" |
| $releaseAll | **[optional]** **True**: release all currently defined matrices from memory; **False**: leave any existing matrices untouched |

## Return Value

Success: True

Failure: results in a fatal error

## Remarks

Repeat calls during a single session will first close the existing dll handle (if open), and if boolean $releaseAll = **True** (the default), also release any existing matrices from memory. The newly-opened dll is either ***EigenDense_Debug.dll*** (in Debug mode) or ***EigenDense.dll*** (in Computing mode). You can switch mode by calling **_Eigen_Debugmode_On()/Off()**, which itself calls **_Eigen_StartUp**().

The default matrix type at the very first initialisation is real float. Thereafter, calling **_Eigen_StartUp**() without any parameters re-initialises the work environment with the current contents of **$EIGEN_MATRIXTYPE**.

The new global settings can be listed by calling **_Eigen_Show_EnvironmentVars()**.

No @error is generated by this call; *any* failure to properly initialise the environment is **fatal**. For example, if global variable **$EIGEN_DLLPATH**, was not user-defined after installing the software, the environment will be unable to find the dlls to hook into; a dedicated message will then remind you to take care of this first.

## Related

*CleanUp*(), *DebugMode_Off*(), *DebugMode_On*(), *SetMatrixType*(), *Show_EnvironmentVars*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_EnvironmentVars()

_Eigen_CleanUp()
```

Test_EigenAssert

Function Reference

# _Eigen_Test_EigenAssert

Trigger an Eigen assert failure message in Debug Mode

```
#include <Eigen4AutoIt.au3>
_Eigen_Test_EigenAssert ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Use this function to test whether Eigen's assert error messages are currently enabled and functional.

This function will produce a displayed message *only* if the current work environment is running in **Debug Mode**.

## Related

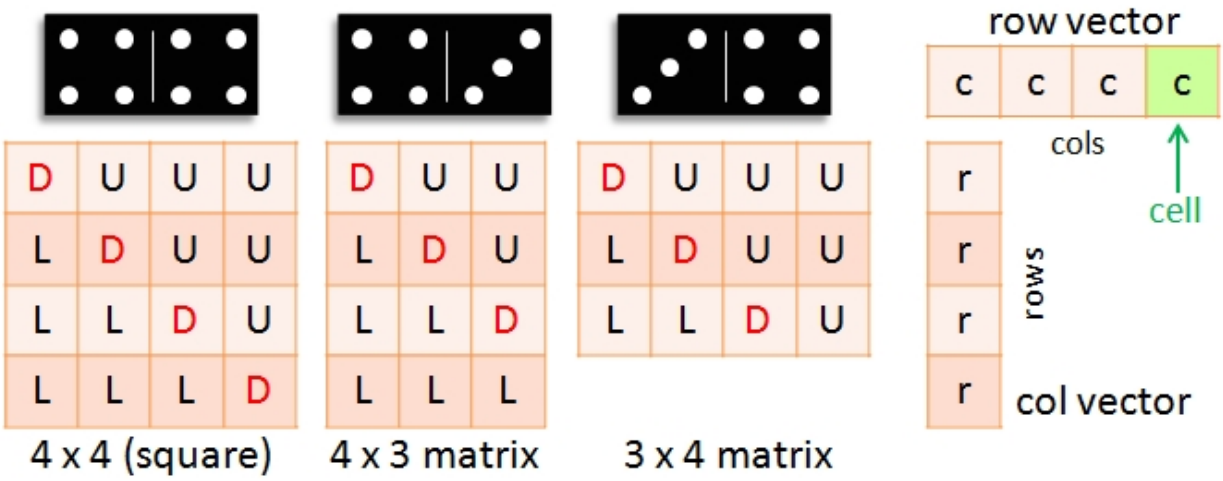
None.

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_SetMatrixType ( "float" )          ; calls _Eigen_Startup()
$matA = _Eigen_CreateMatrix ( 3, 4 )      ; initial type = float

_Eigen_Show_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 4, 3, "int" )    ; warning triggered

_Eigen_Hide_Warnings ()
MsgBox ( 0, "Eigen4AutoIt", "Warnings shown: " & $showWarningMsg )

_Eigen_RedefineMatrix ( $matA, 2, 6, "double" ); NO warning triggered

_Eigen_Cleanup()
```

## Matrix Management

# Matrix Management




A ***matrix*** is a 2D array containing numeric values in each "cell" or "coefficient." Matrix shape is defined by its two dimensions: **rows** and **columns** (or "cols"), **in that order**, illustrated by the "diminoes" at the top of the figure. A ***vector*** can be interpreted as a matrix with one dimension of one (1, or "unity"), and one dimension larger than one. A rowvector has multiple cols, whereas a colvector has multiple rows. Matrices with both dimensions of one are called ***scalars***. You know them well, having used them all your life in ordinary, boring arithmetic.

A matrix is divided by its diagonal (the red **D** cells in the figure) into an (**U**)pper and a (**L**)ower part. Note that in non-square matrices, these parts are different in size, so we cannot just "flip" the upper and lower parts there. This highlights a crucial feature of most matrix operations: **SIZE MATTERS!**

Some operations work only on square matrices; in others involving two or more matrices, the number of *cols* in each first of each pair has to match the number of *rows* in the second one (see Multiplication).

We can create an empty matrix with: **_Eigen_CreateMatrix (** $rows, $cols **)**. New matrices are also created:
1. whenever we create a duplicate with $matB = **_Eigen_CloneMatrix (** $matA **)**, and
2. when a function is called that produces matrix output that is not stored in an existing container pre-supplied by you.

For example:
**_Eigen_Multiply_AB (** $matA, $matB, $matR**=**0 **)** always expects you to supply matrices **A** and **B** to multiply, but leaves it up to you whether or not you provide a container for the result (matrix **R**, which *has* to have the correct dimensions). If you don't, the function will create a new matrix of the right shape and size, and return its ID, so you can store it in a variable.

All **Eigen4AutoIt** functions refer to matrices using these unique **matrix ID**s. These are indices to internal array **$MatrixList**, where all matrix structs and their specs are stored. To facilitate memory management, you can record the current top of this list with **_Eigen_SetListMarker**, and after finishing part of your work, release all matrices created after placing your marker, by calling **_Eigen_ReleaseFromMarker** or **_Eigen_ReleaseFromMarkerExcept**. To release individual matrices, call **_Eigen_ReleaseMatrix(** $matID **)**. The associated matrix ID is then no longer used in the current session (matrices are only ever given an ID *once*). When a matrix is type-converted, a new matrix is created for storing the new type, and the original is also kept, e.g.: **_Eigen_ConvertMatrix_Todouble()**. Generally, whenever functions refer to **ABC**..., this

usually implies *different* matrices (to be parsed by their respective IDs). In a few exceptional cases, you are allowed to parse the same matrix ID twice, but always check first whether a separate function referring to **AA** (instead of **AB**) exists, and use that, to avoid potential aliasing issues.





# Matrix Creation

- CreateMatrix
- CreateMatrix_Constant
- CreateMatrix_FromA
- CreateMatrix_FromA_Transposed
- CreateMatrix_FromABcols
- CreateMatrix_FromABcols_Transposed
- CreateMatrix_FromAblock
- CreateMatrix_FromAblock_Tranposed
- CreateMatrix_FromABrows
- CreateMatrix_FromABrows_Transposed
- CreateMatrix_FromAcols
- CreateMatrix_FromAcols_Transposed
- CreateMatrix_FromArows
- CreateMatrix_FromArows_Transposed
- CreateMatrix_Identity
- CreateMatrix_LinSpaced_ColMajor
- CreateMatrix_LinSpaced_RowMajor
- CreateMatrix_Ones
- CreateMatrix_Random
- CreateMatrix_Zero
- CreateOutput_FromDims
- CreateOutput_FromIDs

See also in main section **Matrix Management**:

- CreateArray_FromMatrix
- CreateMatrix_FromArray
- SplitMatrix_FromAcol
- SplitMatrix_FromAcol_Transposed
- SplitMatrix_FromArow
- SplitMatrix_FromArow_Transposed



## CreateMatrix

Function Reference

# _Eigen_CreateMatrix

Allocate a new matrix variable of specified dimensions in memory

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix ( $rows, $cols[, $matrixType = $EIGEN_MATRIXTYPE ] )
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

Success: new matrix ID (index to $MatrixList of the new entry)

Failure: False, and sets the @error flag to non-zero.

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

To add an externally created matrix struct to the $MatrixList, use *Add_ExistingMatrix()* instead.

## Related

*CloneArray()*, *CloneMatrix()*, *CreateArray_FromMatrix()*, *CreateMatrix_FromArray()*, *RedefineMatrix()*, *RedimExistingMatrix()*, *ReleaseMatrix()*, *Add_ExistingMatrix()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
_Eigen_Show_MatrixList()

$matA = _Eigen_CreateMatrix ( 1, 2 )
```

```
$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseMatrix( $matA )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()
_Eigen_Show_MatrixList()
```





# _Eigen_CreateMatrix_Constant

Allocate a new matrix variable of specified dimensions in memory, fill all cells with supplied value

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_Constant ( $rows, $cols, $value[, $matrixType =
$EIGEN_MATRIXTYPE] )
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $value | constant to fill all matrix cells with |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

| | |
|---|---|
| Success: | new matrix ID (index to $MatrixList of the new entry) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the

array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

## Related

*CreateMatrix*(), *CreateMatrix_Identity*(), *CreateMatrix_Ones*(), *CreateMatrix_Random*(), *CreateMatrix_Zero*(), *CreateMatrix_LinSpaced_ColMajor*(), *CreateMatrix_LinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Constant ( 4, 4, 123 )     ; fill with value: 123
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_FromA

Create a matrix from multiple copies of the source matrix values, tiled horizontally, vertically, or both

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromA ( $matA[, $horizontal_times = 1[,
$vertical_times = 1[, $matR = 0 ]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $horizontal_times | **[optional]** number of duplicates to produce horizontally (minimum: 1) |
| $vertical_times | **[optional]** number of duplicates to produce vertically (minimum: 1) |
| $matR | matrix ID of the destination matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

This function "tiles" the destination matrix with multiple copies of the source matrix values. If the function is called without replication parameters, the input matrix is simply copied.

Results matrix **R** does not need to exist yet, but if it does, it has to fit the duplicated tiled values of matrix **A** exactly.

The replication factors should be positive integers. A duplication value of unity (1) means no additional copies are to be made in the designated orientation.

## Related

*Copy_A_ToB()*, *CreateMatrix_FromA_Transposed()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )

$matB = _Eigen_CreateMatrix_FromA ( $matA, 2, 3 )
_MatrixDisplay ( $matB, "after tiling" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_FromA_Transposed

Create a matrix from multiple transposed copies of the source matrix values, tiled horizontally, vertically, or both

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromA_Transposed ( $matA[, $horizontal_times = 1[,
$vertical_times = 1[, $matR = 0 ]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $horizontal_times | **[optional]** number of duplicates to produce horizontally (minimum: 1) |

| | |
|---|---|
| $vertical_times | **[optional]** number of duplicates to produce vertically (minimum: 1) |
| $matR | matrix ID of the destination matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

This function "tiles" the destination matrix with multiple copies of the transposed source matrix values. If the function is called without replication parameters, the input matrix is simply copied transposed.

Results matrix **R** does not need to exist yet, but if it does, it has to fit the duplicated tiled values of matrix **A** exactly.

The replication factors should be positive integers. A duplication value of unity (1) means no additional copies are to be made in the designated orientation.

## Related

*Copy_A_ToB()*, *CreateMatrix_FromA()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )

$matB = _Eigen_CreateMatrix_FromA_Transposed ( $matA, 2, 3 )
_MatrixDisplay ( $matB, "after tiling" )

_Eigen_CleanUp()
```



### CreateMatrix_FromABcols

Function Reference

# _Eigen_CreateMatrix_FromABcols

Create a new matrix from all columns of matrix A, followed by all columns of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromABcols ( $matA, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first source matrix |
| $matB | matrix ID of the second source matrix |
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

Source matrices **A** and **B** need to have the same number of rows.

Destination matrix **R** is filled with all columns of matrix **A**, followed by all columns of matrix **B**.

## Related

*CreateMatrix_FromABcols_Transposed*(), *CreateMatrix_FromABrows*(), *CreateMatrix_FromABrows_Transposed*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 20 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 101, 120 )
_MatrixDisplay ( $matB, "matrix B" )


$matR = _Eigen_CreateMatrix_FromABcols ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABcols_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

_Eigen_CleanUp()
```

CreateMatrix_FromABcols_Transposed

Function Reference

# _Eigen_CreateMatrix_FromABcols_Transposed

Create a new matrix from all columns of matrix A, followed by all columns of matrix B, transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromABcols_Transposed ( $matA, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first source matrix |
| $matB | matrix ID of the second source matrix |
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

Source matrices **A** and **B** need to have the same number of rows.

Destination matrix **R** is filled with all columns of matrix **A**, followed by all columns of matrix **B**, transposed.

## Related

*CreateMatrix_FromABcols*(), *CreateMatrix_FromABrows*(), *CreateMatrix_FromABrows_Transposed*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 20 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
$matB = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 101, 120 )
_MatrixDisplay ( $matB, "matrix B" )


$matR = _Eigen_CreateMatrix_FromABcols ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABcols_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_FromAblock

Create a new matrix from an existing block in matrix A, including its contents

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromAblock ( $matA, $startRow, $startCol, $blockRows,
$blockCols[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $startRow | row ID (base-0) of the topmost row of the block in source matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in source matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*CreateMatrix_FromAblock_Transposed()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_CreateMatrix_FromAblock ( $matA, 0, 0, 2, 3 )
_MatrixDisplay ( $matR, "block subset" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_FromAblock_Transposed

Create a new matrix from an existing block in matrix A, including its contents, transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromAblock_Transposed ( $matA, $startRow, $startCol,
$blockRows, $blockCols[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $startRow | row ID (base-0) of the topmost row of the block in source matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in source matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |

| | |
|---|---|
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*CreateMatrix_FromAblock()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_CreateMatrix_FromAblock_Transposed ( $matA, 0, 0, 2, 3 )
_MatrixDisplay ( $matR, "block subset" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_FromABrows

Create a new matrix from all rows of matrix A, followed by all rows of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromABrows ( $matA, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first source matrix |
| $matB | matrix ID of the second source matrix |

| | |
|---|---|
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

Source matrices **A** and **B** need to have the same number of columns.

Destination matrix **R** is filled with all rows of matrix **A**, followed by all rows of matrix **B**.

## Related

*CreateMatrix_FromABcols*(), *CreateMatrix_FromABcols_Transposed*(), *CreateMatrix_FromABrows_Transposed*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 20 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 101, 120 )
_MatrixDisplay ( $matB, "matrix B" )


$matR = _Eigen_CreateMatrix_FromABcols ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABcols_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

_Eigen_CleanUp()
```



### CreateMatrix_FromABrows_Transposed

Function Reference

# _Eigen_CreateMatrix_FromABrows_Transposed

Create a new matrix from all rows of matrix A, followed by all rows of matrix B, transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromABrows_Transposed ( $matA, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first source matrix |
| $matB | matrix ID of the second source matrix |
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

| | |
|---|---|
| Success: | new matrix ID |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Source matrices **A** and **B** need to have the same number of columns.

Destination matrix **R** is filled with all rows of matrix **A**, followed by all rows of matrix **B**, transposed.

## Related

*CreateMatrix_FromABcols*(), *CreateMatrix_FromABcols_Transposed*(), *CreateMatrix_FromABrows*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 20 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 101, 120 )
_MatrixDisplay ( $matB, "matrix B" )


$matR = _Eigen_CreateMatrix_FromABcols ( $matA, $matB )
```

```
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABcols_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows ( $matA, $matB )
_MatrixDisplay ( $matR )

$matR = _Eigen_CreateMatrix_FromABrows_Transposed ( $matA, $matB )
_MatrixDisplay ( $matR )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_FromAcols

Create a new matrix from listed columns of matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromAcols ( $matA, $selection[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $selection | string of comma-separated individual column indices and/or index ranges |
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

String $selection should consist of integer index references (base-0) separated by **comma** (accepted alternative separator characters are : "\/;&^()[]{}<>|", i.e., backslash, slash, semi-colon, ampersand, hat, brackets, square brackets, curly brackets, angled brackets, and pipe). Indices can be either individual integers or inclusive index ranges, defined by the first index of the range, followed by **hyphen** ("-", minus sign; accepted alternative range characters are: ":_", i.e., colon and underscore), followed by the last index of the range. References can be in any order; spaces and out-of-bounds indices are ignored. If a range partly exceeds the associated matrix dimension, the valid part is processed.

## Related

*CreateMatrix_FromAcols_Transposed*(), *CreateMatrix_FromArows*(), *CreateMatrix_FromArows_Transposed*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )

$selection = "1,3,7-9"

$matR = _Eigen_CreateMatrix_FromAcols ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromAcols_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

_Eigen_CleanUp()
```



### CreateMatrix_FromAcols_Tranposed

Function Reference

# _Eigen_CreateMatrix_FromAcols_Transposed

Create a new matrix from listed columns of matrix A, transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromAcols_Transposed ( $matA, $selection[, $matR = 0 ]
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $selection | string of comma-separated individual column indices and/or index ranges |

| | |
|---|---|
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

| | |
|---|---|
| Success: | new matrix ID |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

String $selection should consist of integer index references (base-0) separated by **comma** (accepted alternative separator characters are : "\/;&^()[]{}|", i.e., backslash, slash, semi-colon, ampersand, hat, brackets, square brackets, curly brackets, and pipe). Indices can be either individual integers or inclusive index ranges, defined by the first index of the range, followed by **hyphen** ("-", minus sign; accepted alternative range characters are: ":_", i.e., colon and underscore), followed by the last index of the range. References can be in any order; spaces and out-of-bounds indices are ignored. If a range partly exceeds the associated matrix dimension, the valid part is processed.

## Related

*CreateMatrix_FromAcols*(), *CreateMatrix_FromArows*(), *CreateMatrix_FromArows_Transposed*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )

$selection = "1,3,7-9"

$matR = _Eigen_CreateMatrix_FromAcols ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromAcols_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

_Eigen_CleanUp()
```



### CreateMatrix_FromArows

Function Reference

# _Eigen_CreateMatrix_FromArows

Create a new matrix from listed rows of matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromArows ( $matA, $selection[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $selection | string of comma-separated individual column indices and/or index ranges |
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

| | |
|---|---|
| Success: | new matrix ID |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

String $selection should consist of integer index references (base-0) separated by **comma** (accepted alternative separator characters are : "\/;&^()[]{}|", i.e., backslash, slash, semi-colon, ampersand, hat, brackets, square brackets, curly brackets, and pipe). Indices can be either individual integers or inclusive index ranges, defined by the first index of the range, followed by **hyphen** ("-", minus sign; accepted alternative range characters are: ":_", i.e., colon and underscore), followed by the last index of the range. References can be in any order; spaces and out-of-bounds indices are ignored. If a range partly exceeds the associated matrix dimension, the valid part is processed.

## Related

*CreateMatrix_FromAcols*(), *CreateMatrix_FromAcols_Transposed*(), *CreateMatrix_FromArows_Transposed*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )

$selection = "1,3,7-9"

$matR = _Eigen_CreateMatrix_FromAcols ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )
```

```
$matR = _Eigen_CreateMatrix_FromAcols_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_FromArows_Transposed

Create a new matrix from listed rows of matrix A, transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromArows_Transposed ( $matA, $selection[, $matR = 0 ]
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $selection | string of comma-separated individual column indices and/or index ranges |
| $matR | **[optional]** matrix ID of the destination matrix |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

String $selection should consist of integer index references (base-0) separated by **comma** (accepted alternative separator characters are : "\/;&^()[]{}|", i.e., backslash, slash, semi-colon, ampersand, hat, brackets, square brackets, curly brackets, and pipe). Indices can be either individual integers or inclusive index ranges, defined by the first index of the range, followed by **hyphen** ("-", minus sign; accepted alternative range characters are: ":_", i.e., colon and underscore), followed by the last index of the range. References can be in any order; spaces and out-of-bounds indices are ignored. If a range partly exceeds the associated matrix dimension, the valid part is processed.

## Related

*CreateMatrix_FromAcols*(), *CreateMatrix_FromAcols_Transposed*(), *CreateMatrix_FromArows*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )

$selection = "1,3,7-9"

$matR = _Eigen_CreateMatrix_FromAcols ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromAcols_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

$matR = _Eigen_CreateMatrix_FromArows_Transposed ( $matA, $selection )
_MatrixDisplay ( $matR, $selection )

_Eigen_CleanUp()
```



### CreateMatrix_Identity

Function Reference

# _Eigen_CreateMatrix_Identity

Allocate a new matrix variable of specified dimensions in memory, zero all cells, fill diagonal with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_Identity ( $rows, $cols[, $matrixType =
$EIGEN_MATRIXTYPE] )
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases |

| | |
|---|---|
| | are also accepted) |

## Return Value

| | |
|---|---|
| Success: | new matrix ID (index to $MatrixList of the new entry) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

## Related

*CreateMatrix*(), *CreateMatrix_Constant*(), *CreateMatrix_Ones*(), *CreateMatrix_Random*(), *CreateMatrix_Zero*(), *CreateMatrix_LinSpaced_ColMajor*(), *CreateMatrix_LinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Identity ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_LinSpaced_ColMajor

Allocate a new matrix variable of specified dimensions in memory, fill all cells in ColMajor order with succession of equally-spaced values

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_LinSpaced_ColMajor ( $rows, $cols, $firstvalue,
$lastvalue[, $matrixType = $EIGEN_MATRIXTYPE] )
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $firstvalue | starting value of the linearly-spaced sequence to be generated |
| $lastvalue | final value of the linearly-spaced sequence to be generated |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

Success: new matrix ID (index to $MatrixList of the new entry)

Failure: False, and sets the @error flag to non-zero.

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

The total number of matrix elements determines the step size ( ( lastvalue - firstvalue ) / (elements - 1) ).

First and last value cannot be identical.

*ColMajor* order means all rows are first filled in column zero, then in column one, and so forth.

## Related

*CreateMatrix*(), *CreateMatrix_Constant*(), *CreateMatrix_Identity*(), *CreateMatrix_Ones*(), *CreateMatrix_Random*(), *CreateMatrix_Zero*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
$matA = _Eigen_CreateMatrix_LinSpaced_ColMajor ( 3, 4, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_LinSpaced_RowMajor

Allocate a new matrix variable of specified dimensions in memory, fill all cells in RowMajor order with succession of equally-spaced values

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_LinSpaced_RowMajor ( $rows, $cols, $firstvalue,
$lastvalue[, $matrixType = $EIGEN_MATRIXTYPE] )
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $firstvalue | starting value of the linearly-spaced sequence to be generated |
| $lastvalue | final value of the linearly-spaced sequence to be generated |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

Success: new matrix ID (index to $MatrixList of the new entry)

Failure: False, and sets the @error flag to non-zero.

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the

array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

The total number of matrix elements determines the step size ( ( lastvalue - firstvalue ) / (elements - 1) ).

First and last value cannot be identical.

*RowMajor* order means all columns are first filled in row zero, then in row one, and so forth.

## Related

*CreateMatrix*(), *CreateMatrix_Constant*(), *CreateMatrix_Identity*(), *CreateMatrix_Ones*(), *CreateMatrix_Random*(), *CreateMatrix_Zero*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 3, 4, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```



### CreateMatrix_Ones

Function Reference

# _Eigen_CreateMatrix_Ones

Allocate a new matrix variable of specified dimensions in memory, fill all cells with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_Ones ( $rows, $cols[, $matrixType = $EIGEN_MATRIXTYPE]
)
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

Success: new matrix ID (index to $MatrixList of the new entry)

Failure: False, and sets the @error flag to non-zero.

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

## Related

*CreateMatrix*(), *CreateMatrix_Constant*(), *CreateMatrix_Identity*(), *CreateMatrix_Random*(), *CreateMatrix_Zero*(), *CreateMatrix_LinSpaced_ColMajor*(), *CreateMatrix_LinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Ones ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_Random

Allocate a new matrix variable of specified dimensions in memory, fill all cells with random values (0 to ±1)

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_Random ( $rows, $cols[, $matrixType =
$EIGEN_MATRIXTYPE] )
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |

| | |
|---|---|
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

| | |
|---|---|
| Success: | new matrix ID (index to $MatrixList of the new entry) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

## Related

*CreateMatrix*(), *CreateMatrix_Constant*(), *CreateMatrix_Identity*(), *CreateMatrix_Ones*(), *CreateMatrix_Zero*(), *CreateMatrix_LinSpaced_ColMajor*(), *CreateMatrix_LinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_CreateMatrix_Zero

Allocate a new matrix variable of specified dimensions in memory, fill all cells with zeroes

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_Zero ( $rows, $cols[, $matrixType = $EIGEN_MATRIXTYPE]
)
```

## Parameters

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

Success: new matrix ID (index to $MatrixList of the new entry)

Failure: False, and sets the @error flag to non-zero.

## Remarks

To define a vector, set either dimension to unity. **Eigen4AutoIt** does not support one-dimensional matrices.

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. The returned matrix ID is the array index of this new entry. This table "slot" is only ever given out once per **Eigen4AutoIt** session.

## Related

*CreateMatrix*(), *CreateMatrix_Constant*(), *CreateMatrix_Identity*(), *CreateMatrix_Ones*(), *CreateMatrix_Random*()
*CreateMatrix_LinSpaced_ColMajor*(), *CreateMatrix_LinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```



### CreateOutPut_FromDims

Function Reference

# _Eigen_CreateOutput_FromDims

Allocate a new output matrix variable for a given function and input matrix dimensions

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateOutput_FromDims ( $functionName, $rowsA, $colsA[,
$outputMatrix="R"[, $rowsB=0[, $colsB=0[, $rowsC=0[, $colsC=0[, $rowsD=0[,
$colsD=0[, $rowsE=0[, $colsE=0 ]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $functionName | the **Eigen4AutoIt** function for which a new output matrix is to be pre-created (alias wrapper names are accepted too) |
| $rowsA | main input matrix row dimension |
| $colsA | main input matrix column dimension |
| $outputMatrix | **[optional]** the generic matrix letter of the desired output |
| $rowsB | **[optional]** second input matrix row dimension |
| $colsB | **[optional]** second input matrix column dimension |
| $rowsC | **[optional]** third input matrix row dimension |
| $colsC | **[optional]** third input matrix column dimension |
| $rowsD | **[optional]** fourth input matrix row dimension |
| $colsD | **[optional]** fourth input matrix column dimension |
| $rowsE | **[optional]** fifth input matrix row dimension |
| $colsE | **[optional]** fifth input matrix column dimension |

## Return Value

| | |
|---|---|
| Success: | new matrix ID (index in $MatrixList of the new entry) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Normally, calling an **Eigen4AutoIt** function will generate one or multiple new output matrices in which the results of the operation are stored (unless the function has the suffix **_InPlace**). Alternatively, most functions also allow an existing container of the correct dimensions to be pre-supplied as output matrix., in which case its original contents will be overwritten. Calling the current function first will create a new, empty matrix of the correct dimensions and type ahead of time, i.e., before performing the desired operation.

This function is mainly useful in a multi-processing environment, in which one **E4A** process can call this function to create an empty results container,  to have it filled by another **E4A** process.

Several functions require additional information (mostly in the form of boolean flags) to determine an output matrix's dimensions. For example, in JacobiSVD, the $computeThin flag controls the shape of output matrices **U** and **V**. In such cases, this function will throw an error. In other cases such as the EigenSolvers, the $eigenvaluesAsVector flag will simply be assumed to be set to False.

## Related

*CreateMatrix(), CreateOutput_FromIDs*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matR = _Eigen_CreateOutput_FromDims ( "_Eigen_Transpose", 4, 8 )
_MatrixDisplay ( $matR, "Output matrix R, given input matrix [4,8]" )   ;
matR[8,4]

_Eigen_CleanUp()
```





# _Eigen_CreateOutput_FromIDs

Allocate a new output matrix variable for a given function and input matrix IDs

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateOutput_FromIDs ( $functionName, $matA[, $outputMatrix="R"[,
$matB=0[, $matC=0[, $matD=0[, $matE=0 ]]]]] )
```

## Parameters

| | |
|---|---|
| $functionName | the **Eigen4AutoIt** function for which a new output matrix is to be pre-created (alias wrapper names are accepted too) |
| $matA | main input matrix ID |
| $outputMatrix | **[optional]** the generic matrix letter of the desired output |
| $matB | **[optional]** second input matrix ID |
| $matC | **[optional]** fourth input matrix ID |
| $matD | **[optional]** fourth input matrix ID |
| $matE | **[optional]** fifth input matrix ID |

## Return Value

| | |
|---|---|
| Success: | new matrix ID (index in $MatrixList of the new entry) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Normally, calling an **Eigen4AutoIt** function will generate one or multiple new output matrices in which the results of the operation are stored (unless the function has the suffix **_InPlace**). Alternatively, most functions also allow an existing container of the correct dimensions to be pre-supplied as output matrix., in which case its original contents will be overwritten. Calling the current function first will create a new, empty matrix of the correct dimensions and type ahead of time, i.e., before performing the desired operation.

This function is mainly useful in a multi-processing environment, in which one **E4A** process can call this function to create an empty results container,  to have it filled by another **E4A** process.

Several functions require additional information (mostly in the form of boolean flags) to determine an output matrix's dimensions. For example, in JacobiSVD, the $computeThin flag controls the shape of output matrices **U** and **V**. In such cases, this function will throw an error. In other cases such as the EigenSolvers, the $eigenvaluesAsVector flag will simply be assumed to be set to False.

## Related

*CreateMatrix(), CreateOutput_FromDims*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 8 )
$matR = _Eigen_CreateOutput_FromIDs ( "_Eigen_Transpose", $matA )
_MatrixDisplay ( $matR, "Output matrix R, given input matA[4,8]" )      ;
matR[8,4]

_Eigen_CleanUp()
```





# _Eigen_Add_ExistingMatrix

Add an externally-created matrix struct to the MatrixList

```
#include <Eigen4AutoIt.au3>
_Eigen_Add_ExistingMatrix ( $struct, $rows, $cols[, $matrixType =
$EIGEN_MATRIXTYPE] )
```

## Parameters

| | |
|---|---|
| $struct | externally created matrix struct |

| | |
|---|---|
| $rows | matrix row dimension |
| $cols | matrix column dimension |
| $matrixtype | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" (various aliases are also accepted) |

## Return Value

| | |
|---|---|
| Success: | new matrix ID |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

**WARNING**: do **NOT** use this function to create new matrices from scratch; use _Eigen_CreateMatrix() instead. This function is mainly useful in multi-processing environments where matrices may be shared among different clients. In this case, it is the responsibility of the calling programme to provide adequate safeguards to prevent write-access conflicts. For anapplication example, see Tutorial Pooled Multi-Processing.

The external struct needs to conform to **Eigen4AutoIt**'s matrix format requirements in terms of size, type, and ColMajor storage., It also has to have a unique pointer address within the MatrixList, referring to memory owned by the current process.  If the new matrix is of a complex type, its real and imaginary values are expected to be interleaved per matrix cell.

## Related

*RedefineMatrix*(), *Redim_ExistingMatrixFile*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$struct = DllstructCreate ( "float[16]" ) ; please do NOT create new matrices
this way!
_Eigen_Add_ExistingMatrix ( $struct, 4, 4 )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()
```





# _Eigen_CloneArray

Duplicate an existing array, and optionally copy its data too

```
#include <Eigen4AutoIt.au3>
_Eigen_CloneArray ( ByRef $array_src, ByRef $array_dst[, $copyData = True]
)
```

## Parameters

| | |
|---|---|
| $array_src | source array |
| $array_dst | destination array |
| $copyData | **[optional]** **True**: copy array contents too; **False**: create empty, same-sized array duplicate |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

To minimise data traffic, the destination array is parsed *ByRef*, so has to exist already (as a variable, not necessarily as a correctly-sized array).

To duplicate an entire array (including all contents) to a non-declared variable, use the AutoIt command: $arrayNew = $array.

## Related

*CloneMatrix*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *CreateMatrix_FromArray*(), *RedefineMatrix*(), *RedimExistingMatrix*(), *ReleaseMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

Local $arrayA[3][3] = [[1,2,3], [4,5,6], [7,8,9]]
_ArrayDisplay ( $arrayA, "array A" )

$arrayB = _Eigen_CloneArray( $arrayA )
_ArrayDisplay ( $arrayB, "cloned array B" )

_Eigen_CleanUp()
```



### CloneMatrix

Function Reference

# _Eigen_CloneMatrix

Duplicate an existing matrix, and optionally copy its data too

```
#include <Eigen4AutoIt.au3>
_Eigen_CloneMatrix ( $matA[, $copyData = True] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $copyData | **[optional]** **True**: copy matrix contents too; **False**: create empty, same-sized matrix duplicate |

## Return Value

Success: new matrix ID

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*CloneArray*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *CreateMatrix_FromArray*(), *RedefineMatrix*(), *RedimExistingMatrix*(), *ReleaseMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor( $matA, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CloneMatrix( $matA )
_MatrixDisplay ( $matA, "cloned matrix B" )

_Eigen_CleanUp()
```

# _Eigen_CreateMatrix_FromArray

Create a new matrix from an existing array, and optionally copy all array data too

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateMatrix_FromArray ( ByRef $array[, $copyData = True[,
$matrixType = $EIGEN_MATRIXTYPE[, $convertScientificNotation = False[,
$imag = False ]]]] )
```

## Parameters

| | |
|---|---|
| $array | source array |
| $copyData | **[optional]** **True**: copy array contents too; **False**: create empty, same-sized array duplicate |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" |
| $convertScientificNotation | **[optional]** **True**: convert any scientific notation strings to values; **False**: copy as-is |
| $imag | **[optional]** **True**: access imaginary part of a complex matrix; **False**: access real part |

## Return Value

| | |
|---|---|
| Success: | new matrix ID |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Global variable **$EIGEN_MATRIXTYPE** (a string) is (re)defined when **Eigen4AutoIt**'s work environment is (re)initialised. The default is for new matrices to conform to the current environment's matrix type. Only in exceptional circumstances should you explicitly define the matrixtype otherwise.

By default, the array is assumed to contain numeric values. If, however, scientific notation strings may (also) be present, these can be detected and converted (slower). Any remaining text string is stored in a matrix as the value zero.

Complex variants with suffix _Real/_Imag are available; calling the main function on a complex matrix accesses the real part, unless $imag = True (default: False).

## Related

*CloneArray*(), *CloneMatrix*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *RedefineMatrix*(), *RedimExistingMatrix*(), *ReleaseMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

Local $arrayA[3][3] = [[1,2,3], [4,5,6], [7,8,9]]
$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_CreateArray_FromMatrix

Create a new array from an existing matrix, and optionally copy all matrix data too

```
#include <Eigen4AutoIt.au3>
_Eigen_CreateArray_FromMatrix ( $matA, ByRef $array[, $copyData = True[,
$convertScientificNotation = False[, $imag = False ]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix act upon |
| $array | destination array |
| $copyData | **[optional]** **True**: copy array contents too; **False**: create empty, same-sized array duplicate |
| $convertScientificNotation | **[optional]** **True**: convert values to scientific notation strings; **False**: copy as-is (faster) |
| $imag | **[optional]** **True**: access imaginary part of a complex matrix; **False**: access real part |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

To minimise data traffic, the destination array is parsed *ByRef*, so has to exist already (as a variable, not necessarily as a correctly-sized array).

By default, the array is assumed to be filled with numeric values. Alternatively, values may be converted to scientific notation strings at the current precision level (**$EIGEN_DECIMALS**).

Complex variants with suffix _Real/_Imag are available; calling the main function on a complex matrix accesses the real part, unless $imag = True (default: False).

## Related

*CloneArray*(), *CloneMatrix*(), *CreateMatrix*(), *CreateMatrix_FromArray*(), *RedefineMatrix*(), *RedimExistingMatrix*(), *ReleaseMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor( $matA, 1, 12 )

Local $arrayA     ; needs to be declared first
_Eigen_CreateArray_FromMatrix( $matA, $arrayA )
_ArrayDisplay ( $arrayA, "array A" )

_Eigen_CleanUp()
```



### GetActiveMatrix

Function Reference

# _Eigen_GetActiveMatrix

Retrieve the matrix ID for a generic matrix output, identified by letter (A-Z)

```
#include <Eigen4AutoIt.au3>
_Eigen_SetActiveMatrix ($matrixString )
```

## Parameters

| | |
|---|---|
| $matrixString | single matrix letter A-Z (as string) |

## Return Value

| | |
|---|---|
| Success: | matrix ID of the queried active matrix |
| Failure: | -1, but @error = 0 |

## Remarks

Global variable **$EIGEN_ACTIVE_MATRICES** contains bit flags for generic matrix letters A-Z (bit 0-25) Whenever **_Eigen_SetActiveMatrix()** sets any of these prior to a decomposition or statistics call, a matrix ID for each generic output is associated with each *set* flag when that call is made. Immediately afterwards, those matrix IDs can be retrieved **individually** through repeated calls of **_Eigen_GetActiveMatrix("X")**, where "X" is any single letter A-Z (parsed as string).

## Related

*SetActiveMatrix*(), *ResetActiveMatrix*(), *ResetActiveMatrix_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "HQ", True )     ; True = reset all flags first, then set flags for H and Q
_Eigen_Decomp_Hessenberg ( $matA )

$matH = _Eigen_GetActiveMatrix ( "H" )    ; collect new matrix IDs one at a time
$matQ = _Eigen_GetActiveMatrix ( "Q" )

_MatrixDisplay ( $matH, "Hessenberg matrix" )
_MatrixDisplay ( $matQ, "Hessenberg Q matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_GetMatrixCols

Return matrix column dimension

```
#include <Eigen4AutoIt.au3>
_Eigen_GetMatrixCols ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

| | |
|---|---|
| Success: | column dimension of the specified matrix |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

None.

## Related

*GetMatrixRows*(), *GetMatrixType*(), *GetMatrixTypeID*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

$rows = _Eigen_GetMatrixRows ( $matA )
$cols = _Eigen_GetMatrixCols ( $matA )
$type = _Eigen_GetMatrixType ( $matA )
$typeID = _Eigen_GetMatrixTypeID ( $matA )

MsgBox ( 0, "matrix A", "Rows: " & $rows & @CR & "Columns: " & $cols & @CR &
_
     "Type: " & $type & @CR & "Type ID: " & $typeID )

_Eigen_CleanUp()
```





# _Eigen_GetMatrixRows

Return matrix row dimension

```
#include <Eigen4AutoIt.au3>
_Eigen_GetMatrixRows ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

| | |
|---|---|
| Success: | row dimension of the specified matrix |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

None.

## Related

*GetMatrixCols*(), *GetMatrixType*(), *GetMatrixTypeID*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

$rows = _Eigen_GetMatrixRows ( $matA )
$cols = _Eigen_GetMatrixCols ( $matA )
$type = _Eigen_GetMatrixType ( $matA )
$typeID = _Eigen_GetMatrixTypeID ( $matA )

MsgBox ( 0, "matrix A", "Rows: " & $rows & @CR & "Columns: " & $cols & @CR &
_
     "Type: " & $type & @CR & "Type ID: " & $typeID)

_Eigen_CleanUp()
```





# _Eigen_GetMatrixType

Return matrix variable type as string

```
#include <Eigen4AutoIt.au3>
_Eigen_GetMatrixType ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

| | |
|---|---|
| Success: | matrix type (string) of the specified matrix |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The matrix type determines the precision with which values are stored, and, consequently, also the allocated size of the matrix. Type ID (integer) and associated Matrix Type (string) are tabulated here.

## Related

*GetMatrixCols*(), *GetMatrixRows*(), *GetMatrixTypeID*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

$rows = _Eigen_GetMatrixRows ( $matA )
$cols = _Eigen_GetMatrixCols ( $matA )
$type = _Eigen_GetMatrixType ( $matA )
$typeID = _Eigen_GetMatrixTypeID ( $matA )

MsgBox ( 0, "matrix A", "Rows: " & $rows & @CR & "Columns: " & $cols & @CR &
_
     "Type: " & $type & @CR & "Type ID: " & $typeID)

_Eigen_CleanUp()
```





# _Eigen_GetMatrixTypeID

Return matrix variable type as integer ID

```
#include <Eigen4AutoIt.au3>
_Eigen_GetMatrixTypeID ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

| | |
|---|---|
| Success: | variable type (integer ID) of the specified matrix |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Type ID (integer) and associated Matrix Type (string) are tabulated here.

## Related

*GetMatrixCols*(), *GetMatrixRows*(), *GetMatrixType*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

$rows = _Eigen_GetMatrixRows ( $matA )
$cols = _Eigen_GetMatrixCols ( $matA )
$type = _Eigen_GetMatrixType ( $matA )
$typeID = _Eigen_GetMatrixTypeID ( $matA )

MsgBox ( 0, "matrix A", "Rows: " & $rows & @CR & "Columns: " & $cols & @CR &
_
     "Type: " & $type & @CR & "Type ID: " & $typeID)

_Eigen_CleanUp()
```





# _Eigen_IsActiveMatrix

Retrieve the active/inactive state for a generic matrix output, identified by letter (A-Z)

```
#include <Eigen4AutoIt.au3>
_Eigen_IsActiveMatrix ( $matrixstring )
```

## Parameters

| | |
|---|---|
| $matrixString | single matrix letter A-Z (as string) |

## Return Value

Success: True / False, reflecting the state of the associated bit flag

Failure: -1, and sets the @error flag to non-zero.

## Remarks

Unlike most other functions, matrix decompositions can optionally output a multitude of different matrices based on a single set of inputs. Global variable **$EIGEN_ACTIVE_MATRICES** contains bit flags for these generic matrix outputs A-Z (bits 0-25), associated with all decomposition calls and some statistics calls. Before executing such a call, _Eigen_SetActiveMatrix() is used to identify which of the possible inputs and outputs are currently to be evaluated (and created if appropriate containers are not pre-supplied). Use _Eigen_IsActiveMatrix() to test and return the current active/inactive state of such a container.

## Related

*GetActiveMatrix*(), *ResetActiveMatrix*(), *ResetActiveMatrix_Single*(), *SetActiveMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "HQ", True )     ; True = reset all flags first
_Eigen_Decomp_Hessenberg ( $matA )

$matH = _Eigen_GetActiveMatrix ( "H" )    ; collect new IDs one at a time
$matQ = _Eigen_GetActiveMatrix ( "Q" )

_MatrixDisplay ( $matH, "Hessenberg matrix" )
_MatrixDisplay ( $matQ, "Hessenberg Q matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_MatrixDisplay (_MatrixDisplay)

Display a matrix in a ListView

```
 #include <Eigen4AutoIt.au3>
_Eigen_MatrixDisplay ( $matA [, $sTitle = "MatrixDisplay" [, $sMatrixRange
= "" [, $iFlags = 0 [, $vUser_Separator = Default [, $sHeader = Default [,
$iMax_ColWidth = Default [, $iAlt_Color = Default [, $hUser_Function =
""[, $bImag = False ]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to display |
| $sTitle | **[optional]** Title for dialog. Default = "MatrixDisplay". |
| $sMatrixRange | **[optional]** Range of rows/columns to display. Default = entire matrix. (See below for details) |
| $iFlags | **[optional]** Determine UDF options. Add required values together - default = 0<br>1 - Transposes the matrix<br>2 (right) or 4 (center) - Column text alignment (default = left)<br>8 - Verbose = display MsgBox on error and splash screens during processing of large arrays<br>16 - Only 'Copy' buttons displayed<br>32 - No buttons displayed<br>64 - No 'Row' column displayed |
| $vUser_Separator | **[optional]** Sets column display option when copying data to clipboard.<br>Character = Delimiter between columns.<br>Number = Fixed column width - longer items will be truncated.<br>Default = Current separator character (usually "\|"). |
| $sHeader | **[optional]** Column names in header (string of names separated by current separator character - usually "\|"). Default see Remarks. |
| $iMax_Colwidth | **[optional]** Max width to which a ListView column will expand to show content. Default = 350 pixels. |
| $iAlt_Color | **[optional]** ListView alternate rows set to defined color. Default = all rows ListView background color. |
| $hUser_Function | **[optional]** A variable assigned to the user defined function to run. Default = none. See remarks. |
| $bImag | **[optional]** **True**: access imaginary part of a complex matrix; **False**: access real part |

## Return Value

| | |
|---|---|
| Success: | 1 |
| Failure: | 0 and @error flag set as follows: |
| @error: | 1 - $matA is not a matrix |

## Remarks

This **external** function (in ***MatrixDisplay.au3***, included in ***Eigen4AutoIt.au3***) is one of the few Eigen-related functions that does not require the "_Eigen" prefix in its name. It was adapted from _ArrayDisplay()

in ***Array.au3***, one of the standard AutoIt include files. See the Credits section for a listing of authors who contributed to this function and/or to this text. The function is provided separately for copyright reasons, and is formally not part of **Eigen4AutoIt** itself. If needed, you could write your own matrix display function using _Eigen_ReadMatrixValue().

If the function is passed a non-matrix ID variable, the function returns an error and the script continues. If the "verbose" parameter is set in $iFlags (the default), a MsgBox will be displayed that offers the option to exit the script immediately or to continue the script with the normal error return.

Matrix cell values are displayed using the current rounding control (**$EIGEN_DECIMALS**) as upper bound. Integers (even when stored as reals) will be displayed without decimals; scientific notation is used where appropriate (this does not mean that such values are stored as strings; matrix cells can contain values only, including some special cases such as *NaN* ("not-a-number") and *#INF* ("infinity")).

Only 65,525 rows of a matrix can be displayed - this is an AutoIt limitation on the total number of controls that can be displayed in a GUI - and an abitrary display limit of 250 columns has also been set. If the user tries to display a matrix larger than these limits an asterisk ("*") is added to the truncated dimension value which will be displayed in red text. Note that using $iMatrixRange to determine the elements to be displayed allows arrays larger than the display limits to be displayed in sections.

The $sMatrixRange parameter syntax is as follows:

"7" - Show rows 0-7 with all columns
"7:" - Show rows 7-end with all columns
"|7" - Show all rows with columns 0-7
"|7:" - Show all rows with columns 7-end
"7|7" - Show rows 0-7 with columns 0-7
"5:7" - Show rows 5-7 with all columns
"|5:7" - Show all rows with columns 5-7
"7|5:7" - Show rows 0-7 with columns 5-7
"5:7|7" - Show rows 5-7 with columns 0-7
"5:7|5:7" - Show rows 5-7 with columns 5-7

$sHeader names (separated by the current separator character) will be used for as many columns as there are names. If no, or not enough, custom names are specified then the default header of "Row|Col0" for 1D arrays or "Row|Col0|Col1|...|Col n" for 2D is substituted.

The 4 buttons at the bottom of the dialog have the following functions:

- Copy Data & Hdr/Row - copy the matrix or the selected row(s) to the clipboard adding full header and row identification.

- Copy Data Only - copy the matrix or the selected row(s) to the clipboard with no header or row identification.

- Run User Func - run the user-defined function passed in $sUser_Function. This function is entirely separate from the UDF and must be created and coded by the user to accept 2 parameters: the full array and a 1D array holding the selected rows indices with a count in the [0] element. The button is not displayed if no function is specified.

- Exit script - Exit the script immediately.

The $iFlags parameter enables some or all of these buttons to be hidden.

If all buttons are displayed the matrix dimensions are displayed at lower left. They are in red text if any the following occur: row/column truncation, array transposition, or displaying only a range of elements - a

tooltip displays the particular occurence(s).

If the "verbose" parameter is set in $iFlags a splash dialog is displayed during initial processing when the array to display has more than 10,000 elements. A similar dialog is displayed when the "Copy Data" buttons are used on a 10,000+ element display - in this is likely to occur consideration should be given to using a "User Function" to store the matrix for later processing.

**Note**: this help page text was slightly adapted from the AutoIt help page for _ArrayDisplay(), the template UDF for this function.

If a complex matrix ID is supplied as input, the real part is displayed by default; this can be done explicitly by calling _MatrixDisplay_**Real**(). To display the imaginary part instead, please call _MatrixDisplay_**Imag**(). or call _MatrixDisplay with optional parameter $bImag = True.

## Related

None.

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_ReadMatrixValue

Read the contents of a specific matrix cell

```
#include <Eigen4AutoIt.au3>
_Eigen_ReadMatrixValue ($matA, $row, $col[, $imag = False ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $row | row coordinate (base-0) of the cell to act upon |
| $col | column coordinate (base-0) of the cell to act upon |

| | |
|---|---|
| $imag | **[optional]** **True**: access imaginary part of a complex matrix; **False**: access real part |

## Return Value

| | |
|---|---|
| Success: | value in the designated matrix cell |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The value is returned at the current precision level (**$EIGEN_DECIMALS**).

Complex variants with suffix _Real/_Imag are available; calling the main function on a complex matrix accesses the real part, unless $imag = True (default: False).

## Related

*WriteMatrixValue*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetRandom ( $matA )

$row = Random ( 0, 2, 1 )     ; pick random coordinates
$col = Random ( 0, 3, 1 )
$value = _Eigen_ReadMatrixValue ( $matA, $row, $col )

_MatrixDisplay ( $matA, "A(" & $row & "," & $col & ") = " & $value )

_Eigen_CleanUp()
```





# _Eigen_RedefineMatrix

Re-allocate an existing matrix of specified, new dimensions in memory

```
#include <Eigen4AutoIt.au3>
_Eigen_RedefineMatrix ( $matA, $rows, $cols[, $matrixType =
$EIGEN_MATRIXTYPE )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rows | new matrix row dimension |
| $cols | new matrix column dimension |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The matrix has to be currently defined in $MatrixList.

The original struct is released, and replaced with a new one, of the specified dimensions and type.

**All data in the original matrix are lost!**

## Related

*CloneArray*(), *CloneMatrix*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *CreateMatrix_FromArray*(), *RedimExistingMatrix*(), *ReleaseMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_Show_MatrixList()

_Eigen_RedefineMatrix ( $matA, 5, 6 )
_Eigen_Show_MatrixList()

_Eigen_CleanUp()
```



Redim_ExistingMatrix

Function Reference

# _Eigen_Redim_ExistingMatrix

Reorganise the dimensionality of an existing matrix without altering its content

```
#include <Eigen4AutoIt.au3>
_Eigen_Redim_ExistingMatrix ( $matA, $rows, $cols )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rows | new matrix row dimension |
| $cols | new matrix column dimension |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The matrix has to be currently defined in $MatrixList.

The total number of matrix elements (rows multiplied by columns) has to remain the same.

All data in the matrix remain untouched.

## Related

*CloneArray*(), *CloneMatrix*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *CreateMatrix_FromArray*(), *RedefineMatrix*(), *ReleaseMatrix*(), *Redim_ExistingMatrixFile*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 3 )      ; 12 elements
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "before ReDim" )

_Eigen_Redim_ExistingMatrix ( $matA, 3, 4 )     ; still 12 elements
_MatrixDisplay ( $matA, "after ReDim" )

_Eigen_CleanUp()
```



### ReleaseMatrix

Function Reference

# _Eigen_ReleaseMatrix

Release an existing matrix from memory and from the MatrixList

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseMatrix ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: True

Failure: False, but @error = 0

## Remarks

**All data in the released matrix are lost!**

A matrix "variable" is a struct (a contiguous area of 16-byte aligned memory) stored in global array $MatrixList, together with its dimensions, type, and memory address pointer. **_Eigen_ReleaseMatrix()** releases the struct itself from memory, and blanks the slots in $MatrixList, but **does not remove the entry**, which will remain blank for the remainder of the current **Eigen4AutoIt** session (to ensure matrix IDs remain unique, and to avoid confusion; you can overrule this best-practice with _Eigen_RedefineMatrix()).

No @error is generated, because **_Eigen_ReleaseMatrix()** is also used internally on index ranges that may not have proper bounds defined.

## Related

*CloneArray*(), *CloneMatrix*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *CreateMatrix_FromArray*(), *RedefineMatrix*(), *RedimExistingMatrix*(), *ReleaseMatrix_All*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_Startup()
_Eigen_Show_MatrixList()

$matA = _Eigen_CreateMatrix ( 1, 2 )
$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseMatrix ( $matA )
_Eigen_Show_MatrixList()
_Eigen_ReleaseMatrix ( $matB )
_Eigen_Show_MatrixList()
```

```
_Eigen_Cleanup()
```





# _Eigen_ReleaseMatrix_All

Release all existing matrices from memory and reset the MatrixList to zero

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseMatrix_All ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, but @error = 0 |

## Remarks

**All session data are lost!**

Unlike **_Eigen_ReleaseMatrix()**, this function not only releases all existing matrix structs, but also truncates the $MatrixList to zero again, removing all entries permanently.

## Related

*CloneArray*(), *CloneMatrix*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *CreateMatrix_FromArray*(), *RedefineMatrix*(), *RedimExistingMatrix*(), *ReleaseMatrix*(), *ReleaseMatrix_All_Except*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_Startup()
_Eigen_Show_MatrixList()

$matA = _Eigen_CreateMatrix ( 1, 2 )
$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()

_Eigen_ReleaseMatrix ( $matA )
_Eigen_Show_MatrixList()
```

```
_Eigen_ReleaseMatrix ( $matB )
_Eigen_Show_MatrixList()

_Eigen_Cleanup()
```



ReleaseMatrix_All_Except

Function Reference

# _Eigen_ReleaseMatrix_All_Except

Release all existing matrices from memory, with exceptions

```
#include <Eigen4AutoIt.au3>
_Eigen_ReleaseMatrix_All_Except ($mat1 = 0[, $mat2 = 0[, $mat3 = 0[, $mat4
= 0[, $mat5 = 0[, $mat6 = 0[, $mat7 = 0[, $mat8 = 0[, $mat9 = 0[, $mat10 =
0 ]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $mat1 | matrix ID of the (first) matrix to **preserve** |
| $mat# | **[optional]** matrix ID of an additional matrix to **preserve** |

## Return Value

None.

## Remarks

Unlike **_Eigen_ReleaseMatrix_All()**, only individual entries in $MatrixList are processed, with the specified exceptions applied.

## Related

*CloneArray*(), *CloneMatrix*(), *CreateMatrix*(), *CreateArray_FromMatrix*(), *CreateMatrix_FromArray*(), *RedefineMatrix*(), *RedimExistingMatrix*(), *ReleaseMatrix*(), *ReleaseMatrix_All*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_Startup()
_Eigen_Show_MatrixList()

$matA = _Eigen_CreateMatrix ( 1, 2 )
$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_Show_MatrixList()
```

```
_Eigen_ReleaseMatrix ( $matA )
_Eigen_Show_MatrixList()
_Eigen_ReleaseMatrix ( $matB )
_Eigen_Show_MatrixList()

_Eigen_Cleanup()
```





# _Eigen_ResetActiveMatrix

Reset all generic output matrices as inactive

```
#include <Eigen4AutoIt.au3>
_Eigen_ResetActiveMatrix ([$releaseExisting = False ] )
```

## Parameters

| | |
|---|---|
| $releaseExisting | **[optional]** **True**: release from memory all currently associated matrices from prior calls; **False**: leave any associated matrices untouched |

## Return Value

Success: True

Failure: n/a

## Remarks

Resets all bit flags, meaning all generic output matrices are flagged as currently inactive.

Optional boolean $releaseExisting, when True, will, check whether any previous calls have associated a matrix ID with any letter, and if so, release those matrices from memory.

## Related

*GetActiveMatrix*(), *SetActiveMatrix*(), *ResetActiveMatrix_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
```

```
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "HQ", True )     ; True = reset all flags first
_Eigen_Decomp_Hessenberg ( $matA )

$matH = _Eigen_GetActiveMatrix ( "H" )    ; collect new IDs one at a time
$matQ = _Eigen_GetActiveMatrix ( "Q" )

_MatrixDisplay ( $matH, "Hessenberg matrix" )
_MatrixDisplay ( $matQ, "Hessenberg Q matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_ResetActiveMatrix_Single

Reset selected output matrices as inactive

```
#include <Eigen4AutoIt.au3>
_Eigen_ResetActiveMatrix_Single ($matrixString, [$releaseExisting =
False ] )
```

## Parameters

| | |
|---|---|
| $matrixString | (multi-) letter string, any subset of "ABCDEFGHIJKLMNOPQRSTUVWXYZ" (case-insensitive) |
| $releaseExisting | **[optional]** **True**: release from memory any currently associated matrix from prior calls; **False**: leave any associated matrix untouched |

## Return Value

Success: True

Failure: n/a

## Remarks

Resets the bit flags of one or more specific matrices, meaning these are flagged as currently inactive. Any unrecognised matrix letters are ignored.

Optional boolean $releaseExisting, when True, will, check whether any previous calls have associated a matrix ID with any letter, and if so, release those matrices from memory.

## Related

*GetActiveMatrix*(), *SetActiveMatrix*(), *ResetActiveMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "HQ", True )     ; True = reset all flags first
_Eigen_Decomp_Hessenberg ( $matA )

$matH = _Eigen_GetActiveMatrix ( "H" )    ; collect new IDs one at a time
$matQ = _Eigen_GetActiveMatrix ( "Q" )

_MatrixDisplay ( $matH, "Hessenberg matrix" )
_MatrixDisplay ( $matQ, "Hessenberg Q matrix" )

_Eigen_ResetActiveMatrix_Single ( "H" )
_Eigen_ResetActiveMatrix_Single ( "Q" )

_Eigen_CleanUp()
```





# _Eigen_SetActiveMatrix

Flag one or more generic matrix outputs (identified by letter) as active, i.e., to create and/or fill with data

```
#include <Eigen4AutoIt.au3>
_Eigen_SetActiveMatrix ($matrixString[, $resetAll = False[,
$releaseExisting = False ]] )
```

## Parameters

| | |
|---|---|
| $matrixString | (multi-) letter string, any subset of "ABCDEFGHIJKLMNOPQRSTUVWXYZ" (case-insensitive) |
| $resetAll | **[optional]** **True**: reset all bit flags first; **False**: leave any previously set flags untouched |
| $releaseExisting | **[optional]** **True**: release any currently associated matrix from a previous call; **False**: leave any associated matrix untouched |

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | n/a |

## Remarks

Unlike most other functions, matrix decompositions and eigensolvers can optionally output a multitude of different matrices based on a single set of inputs. Global variable **$EIGEN_ACTIVE_MATRICES** contains bit flags for these generic matrix outputs A-Z (bits 0-25), associated with all decomposition calls and some statistics calls. Before executing such a call, _Eigen_SetActiveMatrix() is used to identify which of the possible inputs and outputs are currently to be evaluated (and created if appropriate containers are not pre-supplied). Some outputs may require significantly more processing time than others.

A single call suffices to activate all desired matrices. Enabling non-existent outputs for a particular procedure has no effect, and triggers no errors. If all possible outputs are desired, one can parse special string "all" instead of typing out all letters individually. Adding a <Space> to the letter string sets an extra flag to fill the $DecompSpecs array as well upon return; alternatively, boolean $enableSpecs can be set directly when calling the decomposition.

Multiple calls will add new bit flags to any existing ones set, unless boolean $resetAll = True, in which case all bit flags are reset prior to the new ones being set. Any earlier-generated output matrices remain intact and accessible through their assigned ID.

Optional boolean $releaseExisting, when True, will, for each letter in the parsed list, check whether previous calls have associated a matrix ID with that letter, and if so, release that matrix from memory (its data are then discarded). This is useful if you subsequently call your main decomposition function *without* parsing specific matrix IDs for your output buffers; the wrapper will then create and allocate new matrices as needed, which would increase your memory load repeatedly unless you clean up your earlier allocation(s) beforehand by setting this flag to True.

## Related

*GetActiveMatrix*(), *ResetActiveMatrix*(), *ResetActiveMatrix_Single*(), *IsActiveMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "HQ", True )     ; True = reset all flags first
_Eigen_Decomp_Hessenberg ( $matA )

$matH = _Eigen_GetActiveMatrix ( "H" )    ; collect new IDs one at a time
$matQ = _Eigen_GetActiveMatrix ( "Q" )

_MatrixDisplay ( $matH, "Hessenberg matrix" )
_MatrixDisplay ( $matQ, "Hessenberg Q matrix" )
```

```
_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_SplitMatrix_FromAcol

Split matrix A at a specified column into new matrices B and C

```
#include <Eigen4AutoIt.au3>
_Eigen_SplitMatrix_FromAcol ( $matA, $colindex[, $matB = 0[, $matC = 0 ] ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $colindex | column ID (base-0) of the split in matrix A (start of second destination matrix) |
| $matB | **[optional]** matrix ID of the first destination matrix |
| $matC | **[optional]** matrix ID of the second destination matrix |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

Source matrix **A** needs to have at least two columns.

Destination matrix **B** is filled with columns 0 to ($colindex-1) of matrix **A**.
Destination matrix **C** is filled with the remaining columns of matrix **A**., i.e., $colindex defines the first column of the second destination matrix.

The first and second destination matrices can be retrieved afterwards with _Eigen_GetActiveMatrix ( "B" ) and _Eigen_GetActiveMatrix ( "C" ) respectively.

## Related

*SplitMatrix_FromAcol_Transposed*(), *SplitMatrix_FromArow*(), *SplitMatrix_FromArow_Transposed*()

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SplitMatrix_FromAcol ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromAcol_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_CleanUp()
```





# _Eigen_SplitMatrix_FromAcol_Transposed

Split matrix A at a specified column into new, transposed matrices B and C

```
#include <Eigen4AutoIt.au3>
_Eigen_SplitMatrix_FromAcol_Transposed ( $matA, $colindex[, $matB = 0[, $matC = 0 ] ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $colindex | column ID (base-0) of the split in matrix A (start of second destination matrix) |
| $matB | **[optional]** matrix ID of the first destination matrix |
| $matC | **[optional]** matrix ID of the second destination matrix |

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Source matrix **A** needs to have at least two columns.

Destination matrix **B** is filled with columns 0 to ($colindex-1) of matrix **A**.
Destination matrix **C** is filled with the remaining columns of matrix **A**., i.e., $colindex defines the first column of the second destination matrix.

The first and second destination matrices can be retrieved afterwards with _Eigen_GetActiveMatrix ( "B" ) and _Eigen_GetActiveMatrix ( "C" ) respectively.

## Related

*SplitMatrix_FromAcol()*, *SplitMatrix_FromArow()*, *SplitMatrix_FromArow_Transposed()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SplitMatrix_FromAcol ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromAcol_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_CleanUp()
```

# _Eigen_SplitMatrix_FromArow

Split matrix A at a specified row into new matrices B and C

```
#include <Eigen4AutoIt.au3>
_Eigen_SplitMatrix_FromArow ( $matA, $rowindex[, $matB = 0[, $matC = 0 ] ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $rowindex | row ID (base-0) of the split in matrix A (start of second destination matrix) |
| $matB | **[optional]** matrix ID of the first destination matrix |
| $matC | **[optional]** matrix ID of the second destination matrix |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

Source matrix **A** needs to have at least two rows.

Destination matrix **B** is filled with rows 0 to ($rowindex-1) of matrix **A**.
Destination matrix **C** is filled with the remaining rows of matrix **A**., i.e., $rowindex defines the first rows of the second destination matrix.

The first and second destination matrices can be retrieved afterwards with _Eigen_GetActiveMatrix ( "B" ) and _Eigen_GetActiveMatrix ( "C" ) respectively.

## Related

*SplitMatrix_FromAcol()*, *SplitMatrix_FromAcol_Transposed()*, *SplitMatrix_FromArow_Transposed()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
_Eigen_SplitMatrix_FromAcol ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromAcol_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_CleanUp()
```





# _Eigen_SplitMatrix_FromArow_Transposed

Split matrix A at a specified row into new, transposed matrices B and C

```
#include <Eigen4AutoIt.au3>
_Eigen_SplitMatrix_FromArow_Transposed ( $matA, $rowindex[, $matB = 0[,
$matC = 0 ] ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $rowindex | row ID (base-0) of the split in matrix A (start of second destination matrix) |
| $matB | **[optional]** matrix ID of the first destination matrix |
| $matC | **[optional]** matrix ID of the second destination matrix |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

Source matrix **A** needs to have at least two rows.

Destination matrix **B** is filled with rows 0 to ($rowindex-1) of matrix **A**.
Destination matrix **C** is filled with the remaining rows of matrix **A**., i.e., $rowindex defines the first rows of the second destination matrix.

The first and second destination matrices can be retrieved afterwards with _Eigen_GetActiveMatrix ( "B" ) and _Eigen_GetActiveMatrix ( "C" ) respectively.

## Related

*SplitMatrix_FromAcol*(), *SplitMatrix_FromAcol_Transposed*(), *SplitMatrix_FromArow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 10, 10 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 100 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SplitMatrix_FromAcol ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromAcol_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_SplitMatrix_FromArow_Transposed ( $matA, 6 )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "B" ) )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "C" ) )

_Eigen_CleanUp()
```





# _Eigen_WriteMatrixValue

Write value to a specific matrix cell

```
#include <Eigen4AutoIt.au3>
_Eigen_WriteMatrixValue ($matA, $row, $col, $value[, $imag = False ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $row | row coordinate (base-0) of the cell to act upon |
| $col | column coordinate (base-0) of the cell to act upon |
| $value | the value to store in the designated matrix cell |
| $imag | **[optional]** **True**: access imaginary part of a complex matrix; **False**: access real part |

## Return Value

| | |
|---|---|
| Success: | value in the designated matrix cell, read back from the struct after storing it |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The value is stored at the current precision level (**$EIGEN_DECIMALS**).

Complex variants with suffix _Real/_Imag are available; calling the main function on a complex matrix accesses the real part, unless $imag = True (default: False).

## Related

*ReadMatrixValue()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetRandom ( $matA )

$row = Random ( 0, 2, 1 )     ; pick random coordinates
$col = Random ( 0, 3, 1 )
$value = 1234

_MatrixDisplay ( $matA, "before writing")

_Eigen_WriteMatrixValue ( $matA, $row, $col, $value )

_MatrixDisplay ( $matA, "Written: " & $value )

_Eigen_CleanUp()
```

## File I/O and Type Conversion

# Matrix File I/O and Type Conversion

There are a number of ways to transfer data into, out of, and between matrices.

The simplest case is direct read/write operations on individual matrix cells:

```
_Eigen_WriteMatrixValue ( $matID, $row, $col, $value )
$value = _Eigen_ReadMatrixValue ( $matID, $row, $col )
```

We can also create new matrices from existing (2D, numeric) AutoIt arrays, and vice versa:

```
$array = _Eigen_CreateArrayFromMatrix ( $matID )
$matID = _Eigen_CreateMatrixFromArray ( $array )
```

or transfer all data, ***if*** both matrix *and* array already exist and have the same dimensions:

```
_Eigen_CopyArrayDataToMatrix ( $matID, $array )
_Eigen_CopyMatrixDataToArray ( $matID, $array )
```

Although Eigen itself provides no file I/O whatsoever, **Eigen4AutoIt** does (see below for the format specification):

```
_Eigen_SaveMatrix ( $matID, $filename )
$matID = _Eigen_LoadMatrix ( $filename )
```

If desired, we can specify as third parameter a variable type different from the current work environment to store the data in, for example:

`_Eigen_SaveMatrix ( $matID, $filename, "double" )`, to produce a file twice as big as float (single precision).

Alternatively, we can convert different matrix types in memory too (matrix **B** is created if no ID for it is supplied ($matID = new matrix ID), otherwise $matID = $matB after the call):

```
$matID = _Eigen_ConvertMatrix_AToBint ( $matA, $matB=0 )
$matID = _Eigen_ConvertMatrix_AToBfloat ( $matA, $matB=0 )
$matID = _Eigen_ConvertMatrix_AToBdouble( $matA, $matB=0 )
```

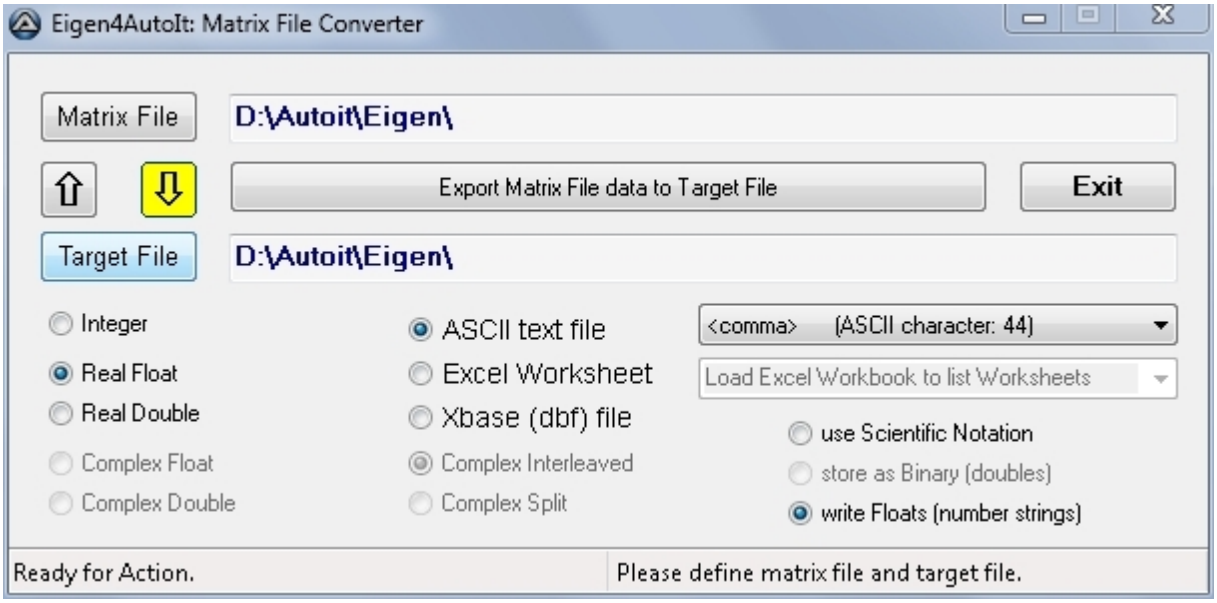
Note that the parameter order is: source, destination; this is almost always the case.

In addition to these internal functions, external utility **MatrixFileConverter** allows the exchange of matrix data in either direction between the native matrix file format and three external formats:

- delimited ASCII
- Excel Worksheets, and
- Xbase dbf files.

Simply define input and output file (the latter need not exist yet), select the direction of conversion (yellow arrow: active), and press the large <Export> button between the two filenames. Various options control the matrix type and the formatting in the external file. Additional options are available by editing settings in

the associated .ini file.




Complex conversions can be confusing, because Eigen's internal storage is in ColMajor order, whereas the supported external formats use RowMajor order (consecutive lines, rows, or records of multiple values per column). Thus if importing from, or exporting to interleaved complex data, the external file is expected to alternate a full row of reals with a full row of imaginary parts, so two consecutive rows form one pair of complex values per column. Alternatively, if complex storage format "Complex Split" is selected, all rows of reals are read/written first, followed by all rows of imaginary parts. Note that in both cases, the external file will have twice as many rows as the complex matrix that is generated from it.
All complex conversions require additional temporary buffer writes, so tend to take much longer than real-only data.

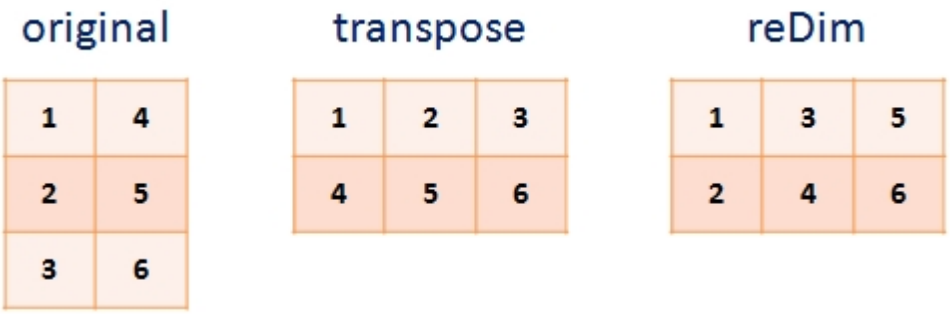
Matrix files are stored in binary form, consisting of a 16-byte header (4 int's) followed by the matrix's raw struct dump, in Colmajor order (so first rows 0-lastrow for column 0, then rows 0-lastrow for column 1, and so on). In the table below, *N* is the total filesize in bytes; the "variable" width of each cell's data depends on the matrix variable type.

| Bytes | Type | Content |
|---|---|---|
| 0 - 3 | <int> | File header: **$EIGEN_MATRIXFILEMARKER** = **202,331,218** |
| 4 - 7 | <int> | File header: number of rows of the matrix |
| 8 - 11 | <int> | File header: number of columns of the matrix |
| 12 - 15 | <int> | File header: variable type ID of the matrix (see Table) |
| 16 - *N* | variable | matrix data, in Colmajor order |

Note that the matrix file marker can be used as BOM (byte-order-marker) in non-standard work environments.

Matrix **type conversions** can be performed in two ways, either acting directly on the file (overwriting the original data if no destination file is provided), or loading the original data from file, and performing the convertion on-the-fly in memory. Note that any conversion to a type that holds less information (<double> to <float> or <int>, <float> to <int>, <complex> to <real>) **destroys information** (accuracy).

A less radical, reversible operation is to **re-dimension** an existing matrix **file**, with **_Eigen_Redim_ExistingMatrixFile()**. In this case, only the second and third integer in the header are rewritten with a different dimensional pair whose product must equal the product of the original pair; the matrix cell data remain unchanged. The same operation can also be performed in memory, by calling **_Eigen_Redim_ExistingMatrix()**. It can be thought of as the counterpart to transposition.








# _Eigen_ConvertMatrix_ToComplexDouble

Convert a matrix of any type to type "complex double" in memory

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrix_ToComplexDouble ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Complex Double" = 16 bytes (not real double, 8 bytes).

If no existing destination matrix of the correct dimensions **and type** is supplied, a new matrix will be created.

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrix_ToFloat*(), *ConvertMatrix_ToInt*(),*ConvertMatrix_ToComplexFloat*(), *ConvertMatrix_ToDouble*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp ( "double" )

$matD = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matD, "original double" )

_Eigen_SetMatrixType ( "complexdouble" )
$matCD = _Eigen_ConvertMatrix_ToComplexDouble ( $matD )
_MatrixDisplay_Real ( $matCD, "double -> " & _Eigen_GetMatrixType
( $matCD ) )
_MatrixDisplay_Imag ( $matCD, "imag part" )

_Eigen_SetMatrixType ( "complexfloat" )
$matCF = _Eigen_ConvertMatrix_ToComplexFloat ( $matD )
_MatrixDisplay_Real ( $matCF, "double -> " & _Eigen_GetMatrixType
( $matCF ) )
_MatrixDisplay_Imag ( $matCD, "imag part" )

_Eigen_CleanUp()
```





# _Eigen_ConvertMatrix_ToComplexFloat

Convert a matrix of any type to type "complex float" in memory

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrix_ToComplexFloat ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Float" = **real** float, 4 bytes (not complex float, 8 bytes).

If no existing destination matrix of the correct dimensions **and type** is supplied, a new matrix will be created.

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrix_ToDouble*(), *ConvertMatrix_ToInt*(),*ConvertMatrix_ToFloat*(), *ConvertMatrix_ToComplexDouble*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp ( "double" )

$matD = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matD, "original double" )

_Eigen_SetMatrixType ( "complexdouble" )
$matCD = _Eigen_ConvertMatrix_ToComplexDouble ( $matD )
_MatrixDisplay_Real ( $matCD, "double -> " & _Eigen_GetMatrixType
( $matCD ) )
_MatrixDisplay_Imag ( $matCD, "imag part" )

_Eigen_SetMatrixType ( "complexfloat" )
$matCF = _Eigen_ConvertMatrix_ToComplexFloat ( $matD )
_MatrixDisplay_Real ( $matCF, "double -> " & _Eigen_GetMatrixType
( $matCF ) )
_MatrixDisplay_Imag ( $matCD, "imag part" )

_Eigen_CleanUp()
```

# _Eigen_ConvertMatrix_ToDouble

Convert a matrix of any type to type "real double" in memory

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrix_ToDouble ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Double" = **real** double, 8 bytes (not complex double, 16 bytes).

If no existing destination matrix of the correct dimensions **and type** is supplied, a new matrix will be created.

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrix_ToFloat*(), *ConvertMatrix_ToInt*(),*ConvertMatrix_ToComplexFloat*(), *ConvertMatrix_ToComplexDouble*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp ( "double" )
```

```
$matD = _Eigen_CreateMatrix_Ones ( 4, 4 )
_Eigen_CwiseScalarOp ( $matD, "+", 0.000001000001 )   ; 6th & 12th decimal set
_MatrixDisplay ( $matD, "original double" )

_Eigen_SetMatrixType ( "int" )
$matD2I = _Eigen_ConvertMatrix_ToInt ( $matD )
_MatrixDisplay ( $matD2I, "double -> " & _Eigen_GetMatrixType ( $matD2I ) )

_Eigen_SetMatrixType ( "float" )
$matD2F = _Eigen_ConvertMatrix_ToFloat ( $matD )
_MatrixDisplay ( $matD2F, "double -> " & _Eigen_GetMatrixType ( $matD2F ) )

_Eigen_SetMatrixType ( "double" )
$matF2D = _Eigen_ConvertMatrix_ToDouble ( $matD2F )
_MatrixDisplay ( $matF2D, "float -> " & _Eigen_GetMatrixType ( $matF2D ) )
; NB the stored float values are not exactly representable in double
precision

_Eigen_CleanUp()
```





# _Eigen_ConvertMatrix_ToFloat

Convert a matrix of any type to type "real float" in memory

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrix_ToFloat ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Float" = **real** float, 4 bytes (not complex float, 8 bytes).

If no existing destination matrix of the correct dimensions **and type** is supplied, a new matrix will be created.

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrix_ToDouble(), ConvertMatrix_ToInt(),ConvertMatrix_ToComplexFloat(), ConvertMatrix_ToComplexDouble()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp ( "double" )

$matD = _Eigen_CreateMatrix_Ones ( 4, 4 )
_Eigen_CwiseScalarOp ( $matD, "+", 0.000001000001 )   ; 6th & 12th decimal set
_MatrixDisplay ( $matD, "original double" )

_Eigen_SetMatrixType ( "int" )
$matD2I = _Eigen_ConvertMatrix_ToInt ( $matD )
_MatrixDisplay ( $matD2I, "double -> " & _Eigen_GetMatrixType ( $matD2I ) )

_Eigen_SetMatrixType ( "float" )
$matD2F = _Eigen_ConvertMatrix_ToFloat ( $matD )
_MatrixDisplay ( $matD2F, "double -> " & _Eigen_GetMatrixType ( $matD2F ) )

_Eigen_SetMatrixType ( "double" )
$matF2D = _Eigen_ConvertMatrix_ToDouble ( $matD2F )
_MatrixDisplay ( $matF2D, "float -> " & _Eigen_GetMatrixType ( $matF2D ) )
; NB the stored float values are not exactly representable in double
precision

_Eigen_CleanUp()
```



ConvertMatrix_ToInt

Function Reference

# _Eigen_ConvertMatrix_ToInt

# ADD EXAMPLE

Convert a matrix of any type to type "int" in memory

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrix_ToInt ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Int" = 4 bytes.

If no existing destination matrix of the correct dimensions **and type** is supplied, a new matrix will be created.

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrix_ToDouble*(), *ConvertMatrix_ToFloat*(),*ConvertMatrix_ToComplexFloat*(), *ConvertMatrix_ToComplexDouble*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp ( "double" )

$matD = _Eigen_CreateMatrix_Ones ( 4, 4 )
_Eigen_CwiseScalarOp ( $matD, "+", 0.000001000001 )   ; 6th & 12th decimal set
_MatrixDisplay ( $matD, "original double" )

_Eigen_SetMatrixType ( "int" )
$matD2I = _Eigen_ConvertMatrix_ToInt ( $matD )
_MatrixDisplay ( $matD2I, "double -> " & _Eigen_GetMatrixType ( $matD2I ) )

_Eigen_SetMatrixType ( "float" )
$matD2F = _Eigen_ConvertMatrix_ToFloat ( $matD )
_MatrixDisplay ( $matD2F, "double -> " & _Eigen_GetMatrixType ( $matD2F ) )
```

```
_Eigen_SetMatrixType ( "double" )
$matF2D = _Eigen_ConvertMatrix_ToDouble ( $matD2F )
_MatrixDisplay ( $matF2D, "float -> " & _Eigen_GetMatrixType ( $matF2D ) )
; NB the stored float values are not exactly representable in double
precision

_Eigen_CleanUp()
```



ConvertMatrixFile_ToComplexDouble

Function Reference

# _Eigen_ConvertMatrixFile_ToComplexDouble

Convert a matrix file of any type to type "complex double"

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrixFile_ToComplexDouble ( $filename_src[, $filename_dst =
"" ] )
```

## Parameters

| | |
|---|---|
| $filename_src | source matrix file |
| $filename_dst | **[optional]** destination matrix file |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Complex Double" = 16 bytes (not real double, 8 bytes).

Any existing destination file will be overwritten.
**If no destination filename is supplied, the source file itself is converted!**

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrixFile_ToFloat*(), *ConvertMatrixFile_ToInt*(), *ConvertMatrixFile_ToDouble*(), *ConvertMatrixFile_ToComplexFloat*()

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp ( "double" )

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matD, "original double" )

$tempfile = _TempFile()
_Eigen_SaveMatrix ( $matD, $tempfile )    ; saved as double

_Eigen_ConvertMatrixFile_ToComplexDouble ( $tempfile )      ; converts source
file
_Eigen_SetMatrixType ( "complexdouble" )
$matCD = _Eigen_LoadMatrix ( $tempfile, 0, "complexdouble" )      ; 0 = no
existing container used
_MatrixDisplay_Real ( $matCD, "double -> " & _Eigen_GetMatrixType
( $matCD ) )
_MatrixDisplay_Imag ( $matCD, "imag part" )
_Eigen_ConvertMatrixFile_ToComplexFloat ( $tempfile )

_Eigen_SetMatrixType ( "complexfloat" )
$matCF = _Eigen_LoadMatrix ( $tempfile, 0, "complexfloat" )
_MatrixDisplay_Real ( $matCF, "complex double -> " & _Eigen_GetMatrixType
( $matCF ) )
_MatrixDisplay_Imag ( $matCF, "imag part" )

FileDelete ( $tempfile )

_Eigen_CleanUp()
```





# _Eigen_ConvertMatrixFile_ToComplexFloat

Convert a matrix file of any type to type "complex float"

```
#include <Eigen4AutoIt.au3>
```

```
_Eigen_ConvertMatrixFile_ToComplexFloat ( $filename_src[, $filename_dst =
"" ] )
```

## Parameters

| | |
|---|---|
| $filename_src | source matrix file |
| $filename_dst | **[optional]** destination matrix file |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

"complex Float" = 8 bytes (not real float, 4 bytes).

Any existing destination file will be overwritten.
**If no destination filename is supplied, the source file is itself converted!**

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrixFile_ToDouble*(), *ConvertMatrixFile_ToInt*(), *ConvertMatrixFile_ToComplexDouble*(), *ConvertMatrixFile_ToFloat*()

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp ( "double" )

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matD, "original double" )

$tempfile = _TempFile()
_Eigen_SaveMatrix ( $matD, $tempfile )    ; saved as double

_Eigen_ConvertMatrixFile_ToComplexDouble ( $tempfile )      ; converts source
file
_Eigen_SetMatrixType ( "complexdouble" )
$matCD = _Eigen_LoadMatrix ( $tempfile, 0, "complexdouble" )      ; 0 = no
existing container used
```

```
_MatrixDisplay_Real ( $matCD, "double -> " & _Eigen_GetMatrixType
( $matCD ) )
_MatrixDisplay_Imag ( $matCD, "imag part" )
_Eigen_ConvertMatrixFile_ToComplexFloat ( $tempfile )

_Eigen_SetMatrixType ( "complexfloat" )
$matCF = _Eigen_LoadMatrix ( $tempfile, 0, "complexfloat" )
_MatrixDisplay_Real ( $matCF, "complex double -> " & _Eigen_GetMatrixType
( $matCF ) )
_MatrixDisplay_Imag ( $matCF, "imag part" )

FileDelete ( $tempfile )

_Eigen_CleanUp()
```





# _Eigen_ConvertMatrixFile_ToDouble

Convert a matrix file of any type to type "real double"

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrixFile_ToDouble ( $filename_src[, $filename_dst = "" ] )
```

## Parameters

| | |
|---|---|
| $filename_src | source matrix file |
| $filename_dst | **[optional]** destination matrix file |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Double" = **real** double, 8 bytes (not complex double, 16 bytes).

Any existing destination file will be overwritten.
**If no destination filename is supplied, the source file itself is converted!**

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!

2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrixFile_ToFloat*(), *ConvertMatrixFile_ToInt*(), *ConvertMatrixFile_ToComplexDouble*(), *ConvertMatrixFile_ToComplexFloat*()

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp ( "double" )

$matA = _Eigen_CreateMatrix_Ones ( 4, 4 )
_Eigen_CwiseScalarOp ( $matD, "+", 0.000001000001 )   ; 6th & 12th decimal set
_MatrixDisplay ( $matD, "original double" )

$tempfile = _TempFile()
_Eigen_SaveMatrix ( $matD, $tempfile )    ; saved as double

_Eigen_ConvertMatrixFile_ToFloat ( $tempfile ) ; converts source file
_Eigen_SetMatrixType ( "float" )
$matF = _Eigen_LoadMatrix ( $tempfile, 0, "float" )   ; 0 = no existing
container used
_MatrixDisplay ( $matF, "double -> " & _Eigen_GetMatrixType ( $matF ) )

_Eigen_ConvertMatrixFile_ToInt ( $tempfile )
_Eigen_SetMatrixType ( "int" )
$matI = _Eigen_LoadMatrix ( $tempfile, 0, "int" )
_MatrixDisplay ( $matI, "float -> " & _Eigen_GetMatrixType ( $matI ) )

_Eigen_ConvertMatrixFile_ToDouble ( $tempfile )
_Eigen_SetMatrixType ( "double" )
$matI2D = _Eigen_LoadMatrix ( $tempfile, 0, "double" )
_MatrixDisplay ( $matI2D, "int -> " & _Eigen_GetMatrixType ( $matI2D ) )

FileDelete ( $tempfile )

_Eigen_CleanUp()
```





# _Eigen_ConvertMatrixFile_ToFloat

Convert a matrix file of any type to type "real float"

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrixFile_ToFloat ( $filename_src[, $filename_dst = "" ] )
```

## Parameters

| | |
|---|---|
| $filename_src | source matrix file |
| $filename_dst | **[optional]** destination matrix file |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Float" = **real** float, 4 bytes (not complex float, 8 bytes).

Any existing destination file will be overwritten.
**If no destination filename is supplied, the source file is itself converted!**

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrixFile_ToDouble*(), *ConvertMatrixFile_ToInt*(), *ConvertMatrixFile_ToComplexDouble*(), *ConvertMatrixFile_ToComplexFloat*()

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp ( "double" )

$matA = _Eigen_CreateMatrix_Ones ( 4, 4 )
_Eigen_CwiseScalarOp ( $matD, "+", 0.000001000001 )   ; 6th & 12th decimal set
_MatrixDisplay ( $matD, "original double" )

$tempfile = _TempFile()
_Eigen_SaveMatrix ( $matD, $tempfile )    ; saved as double

_Eigen_ConvertMatrixFile_ToFloat ( $tempfile ) ; converts source file
_Eigen_SetMatrixType ( "float" )
$matF = _Eigen_LoadMatrix ( $tempfile, 0, "float" )   ; 0 = no existing
container used
_MatrixDisplay ( $matF, "double -> " & _Eigen_GetMatrixType ( $matF ) )

_Eigen_ConvertMatrixFile_ToInt ( $tempfile )
_Eigen_SetMatrixType ( "int" )
```

```
$matI = _Eigen_LoadMatrix ( $tempfile, 0, "int" )
_MatrixDisplay ( $matI, "float -> " & _Eigen_GetMatrixType ( $matI ) )

_Eigen_ConvertMatrixFile_ToDouble ( $tempfile )
_Eigen_SetMatrixType ( "double" )
$matI2D = _Eigen_LoadMatrix ( $tempfile, 0, "double" )
_MatrixDisplay ( $matI2D, "int -> " & _Eigen_GetMatrixType ( $matI2D ) )

FileDelete ( $tempfile )

_Eigen_CleanUp()
```





# _Eigen_ConvertMatrixFile_ToInt

Convert a matrix file of any type to type "int"

```
#include <Eigen4AutoIt.au3>
_Eigen_ConvertMatrixFile_ToInt ( $filename_src[, $filename_dst = "" ] )
```

## Parameters

| | |
|---|---|
| $filename_src | source matrix file |
| $filename_dst | **[optional]** destination matrix file |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

"Int" = 4 bytes.

Any existing destination file will be overwritten.
**If no destination filename is supplied, the source file is itself converted!**

If the destination type holds less information than the source type, you'll lose accuracy in the conversion.

BEWARE:
1) The appearance of cell contents (rounding) is affected by the precision level of the current work environment!
2) Not every real value can be accurately represented in binary, even at double precision!

## Related

*ConvertMatrixFile_ToDouble*(), *ConvertMatrixFile_ToFloat*(), *ConvertMatrixFile_ToComplexDouble*(), *ConvertMatrixFile_ToComplexFloat*()

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp ( "double" )

$matA = _Eigen_CreateMatrix_Ones ( 4, 4 )
_Eigen_CwiseScalarOp ( $matD, "+", 0.000001000001 )   ; 6th & 12th decimal set
_MatrixDisplay ( $matD, "original double" )

$tempfile = _TempFile()
_Eigen_SaveMatrix ( $matD, $tempfile )    ; saved as double

_Eigen_ConvertMatrixFile_ToFloat ( $tempfile ) ; converts source file
_Eigen_SetMatrixType ( "float" )
$matF = _Eigen_LoadMatrix ( $tempfile, 0, "float" )   ; 0 = no existing
container used
_MatrixDisplay ( $matF, "double -> " & _Eigen_GetMatrixType ( $matF ) )

_Eigen_ConvertMatrixFile_ToInt ( $tempfile )
_Eigen_SetMatrixType ( "int" )
$matI = _Eigen_LoadMatrix ( $tempfile, 0, "int" )
_MatrixDisplay ( $matI, "float -> " & _Eigen_GetMatrixType ( $matI ) )

_Eigen_ConvertMatrixFile_ToDouble ( $tempfile )
_Eigen_SetMatrixType ( "double" )
$matI2D = _Eigen_LoadMatrix ( $tempfile, 0, "double" )
_MatrixDisplay ( $matI2D, "int -> " & _Eigen_GetMatrixType ( $matI2D ) )

FileDelete ( $tempfile )

_Eigen_CleanUp()
```





# _Eigen_LoadMatrix

Load a new matrix variable from file

```
#include <Eigen4AutoIt.au3>
_Eigen_LoadMatrix ( $filename[, $matA = 0[, $matrixType =
$EIGEN_MATRIXTYPE ]] )
```

## Parameters

| | |
|---|---|
| $filename | [full path +] filename of the matrix file to act upon |
| $matA | **[optional]** matrix ID of the pre-existing matrix to fill |
| $matrixType | **[optional]** "int", "float" (default), "double", "complexfloat", "complexdouble" |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The matrix file has to conform to **Eigen4AutoIt**'s internal format for saving matrices.

If a pre-existing matrix is supplied as container (rather than creating a new matrix), it has to have the **same** dimensions as specified in the file header.

If the $matrixType is specified and differs from that of the matrix file, a type conversion is carried out. Note that if the specified type holds less information, you'll lose accuracy in the conversion.

## Related

*CreateMatrix*(), *SaveMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 12 )

Local $tempfile = _TempFile()
_Eigen_SaveMatrix ( $matA, $tempfile )
_MatrixDisplay ( $matA, "A saved" )

$matB = _Eigen_LoadMatrix ( $tempfile )
_MatrixDisplay ( $matB, "B loaded" )
FileDelete ( $tempfile )

_Eigen_CleanUp()
```



### Redim_ExistingMatrixFile

Function Reference

# _Eigen_Redim_ExistingMatrixFile

Reorganise the dimensionality of a matrix stored on file, without altering its content

```
#include <Eigen4AutoIt.au3>
_Eigen_Redim_ExistingMatrixFile ( $filename, $rows, $cols )
```

## Parameters

| | |
|---|---|
| $filename | [full path +] filename of the matrix file to act upon |
| $rows | new matrix row dimension |
| $cols | new matrix column dimension |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

The total number of matrix elements (rows multiplied by columns) has to remain the same.

All data in the matrix remain untouched; only the file header is rewritten.

## Related

*CreateMatrix*(), *LoadMatrix*(), *SaveMatrix*(), *Redim_ExistingMatrix*()

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 3 )      ; 12 elements
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "before ReDim" )

Local $tempfile = _TempFile()
_Eigen_SaveMatrix ( $matA, $tempfile )

_Eigen_RedimExistingMatrixFile ( $tempfile, 3, 4 )    ; still 12 elements

$matB = _Eigen_LoadMatrix ( $tempfile )
_MatrixDisplay ( $matB, "after ReDim" )
FileDelete ( $tempfile )
```

```
_Eigen_CleanUp()
```



SaveMatrix

Function Reference

# _Eigen_SaveMatrix

Save an existing matrix variable to file

```
#include <Eigen4AutoIt.au3>
_Eigen_SaveMatrix ( $matA, $filename[, $matrixType = $EIGEN_MATRIXTYPE ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to save |
| $filename | [full path +] filename of the matrix file to store the data |
| $matrixType | **[optional]** "int", "float" (default), "double", "complexfloat", "complexdouble" |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

If the $matrixType is specified and differs from that of the matrix file, a type conversion is carried out. Note that if the specified type holds less information, you'll lose accuracy in the conversion.

The matrix is saved in **Eigen4AutoIt's** internal format.

## Related

*CreateMatrix(), LoadMatrix()*

## Example

```
#include "Eigen4AutoIt.au3"
#include <File.au3>

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 12 )
```

```
Local $tempfile = _TempFile()
_Eigen_SaveMatrix ( $matA, $tempfile )
_MatrixDisplay ( $matA, "A saved" )

$matB = _Eigen_LoadMatrix ( $tempfile )
_MatrixDisplay ( $matB, "B loaded" )
FileDelete ( $tempfile )

_Eigen_CleanUp()
```

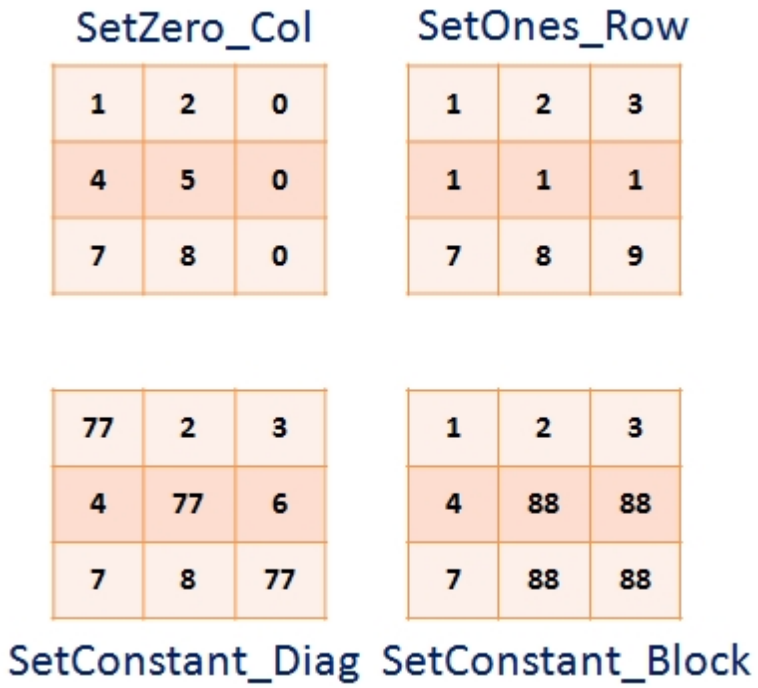


## Set

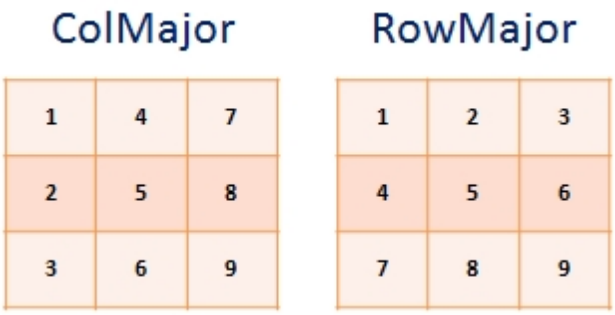
# Set

**Set** fills all cells in a matrix or part thereof with a single value (zero, one, or a user-defined constant), with random values (0 to ±1, sampled from a flat parent distribution, so all values within the range are equally likely), or with linearly-spaced values.  Some examples:


Most **Set** functions can also be applied to a particular matrix part, such as one specific column or row, the diagonal cells, or a rectangular block defined by its top left coordinates, its height, and its width. In the examples below, the target matrix was intitially filled with values 1 through 9, Rowwise (see also below).




A linearly-spaced sequence of values is defined through its first and last value and the number of elements to be filled; note that if the last value is smaller than the first, a negative step is implied. The step size is determined by the number of cells comprising the target area (column, diagonal, row, or full matrix; minimum: 2 cells). If an entire matrix is to be filled with LinSpaced values, the order can be either Colmajor (vertical) or RowMajor (horizontal). SetLinSpaced functions are not supported for complex matrices, but can be applied to their real or imaginary parts separately.






SetConstant

Function Reference

# _Eigen_SetConstant

Fill all matrix cells with the same value

```
#include <Eigen4AutoIt.au3>
```

```
_Eigen_SetConstant ( $matA, $value )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $value | constant to fill all matrix cells with |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetConstant_Block*(), *SetConstant_Col*(), *SetConstant_Diag*(), *SetConstant_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetConstant ( $matA, 123 )

_MatrixDisplay ( $matA, "SetConstant" )

_Eigen_CleanUp()
```





# _Eigen_SetConstant_Block

Fill all matrix cells in a block with the same value

```
#include <Eigen4AutoIt.au3>
_Eigen_SetConstant_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols, $value )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $value | constant to fill all matrix cells inside the block with |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetConstant*(), *SetConstant_Col*(), *SetConstant_Diag*(), *SetConstant_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetZero ( $matA )

$startRow = 1
$startCol = 1
$blockRows = 2
$blockCols = 2
_Eigen_SetConstant_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols, 123 )

_MatrixDisplay ( $matA, "SetConstant_Block" )

_Eigen_CleanUp()
```



### SetConstant_Col

Function Reference

# _Eigen_SetConstant_Col

Fill all matrix cells in one column with the same value

```
#include <Eigen4AutoIt.au3>
_Eigen_SetConstant_Col ( $matA, $colindex, $value )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |
| $value | constant to fill all matrix cells in the designated column with |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetConstant*(), *SetConstant_Block*(), *SetConstant_Diag*(), *SetConstant_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetZero ( $matA )

$colindex = 2
_Eigen_SetConstant_Col ( $matA, $colindex, 123 )

_MatrixDisplay ( $matA, "SetConstant_Col" )

_Eigen_CleanUp()
```

# _Eigen_SetConstant_Diag

Fill all diagonal cells of a matrix with the same value

```
#include <Eigen4AutoIt.au3>
_Eigen_SetConstant_Diag ( $matA, $value )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $value | constant to fill all diagonal matrix cells with |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetConstant*(), *SetConstant_Block*(), *SetConstant_Col*(), *SetConstant_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetZero ( $matA )

_Eigen_SetConstant_Diag ( $matA, 123 )

_MatrixDisplay ( $matA, "SetConstant_Diag" )

_Eigen_CleanUp()
```

# _Eigen_SetConstant_Row

Fill all matrix cells in one row with the same value

```
#include <Eigen4AutoIt.au3>
_Eigen_SetConstant_Row ( $matA, $rowindex, $value )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |
| $value | constant to fill all matrix cells in the designated row with |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetConstant*(), *SetConstant_Block*(), *SetConstant_Col*(), *SetConstant_Diag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetZero ( $matA )

$rowindex = 1
_Eigen_SetConstant_Row ( $matA, $rowindex, 123 )

_MatrixDisplay ( $matA, "SetConstant_Row" )

_Eigen_CleanUp()
```



### SetIdentity

Function Reference

# _Eigen_SetIdentity

Fill a matrix with zeroes, and the diagonal with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_SetIdentity ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

An Identity matrix is the matrix equivalent of a scalar value of unity.

## Related

*SetIdentity_Block*(), *SetOnes_Diag*(), *SetZero*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 5, 5 )
_Eigen_SetIdentity ( $matA )

_MatrixDisplay ( $matA, "SetIdentity" )

_Eigen_CleanUp()
```





# _Eigen_SetIdentity_Block

Fill a matrix block with zeroes, and the block diagonal with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_SetIdentity_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetIdentity*(), *SetOnes_Diag*(), *SetZero_Block*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetConstant ( $matA, 123 )

$startRow = 1
$startCol = 1
$blockRows = 2
$blockCols = 2
_Eigen_SetIdentity_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols )

_MatrixDisplay ( $matA, "Identity_Block" )

_Eigen_CleanUp()
```



### SetLinSpaced_Col

Function Reference

# _Eigen_SetLinSpaced_Col

Fill all matrix cells in one column with a succession of equally-spaced values

```
#include <Eigen4AutoIt.au3>
_Eigen_SetLinSpaced_Col ( $matA, $colindex, $firstvalue, $lastvalue )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |
| $firstvalue | starting value of the linearly-spaced sequence to be generated |
| $lastvalue | final value of the linearly-spaced sequence to be generated |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The total number of rows determines the step size ( ( lastvalue - firstvalue ) / (rows - 1) ).

First and last value cannot be identical.

## Related

*SetLinSpaced_ColMajor*(), *SetLinSpaced_Diag*(), *SetLinSpaced_Row*(), *SetLinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetZero ( $matA )

$colindex = 2
_Eigen_SetLinSpaced_Col ( $matA, $colindex, 1, 4 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```

SetLinSpaced_ColMajor

Function Reference

# _Eigen_SetLinSpaced_ColMajor

Fill all matrix cells in ColMajor order with a succession of equally-spaced values

```
#include <Eigen4AutoIt.au3>
_Eigen_SetLinSpaced_ColMajor ( $matA, $firstvalue, $lastvalue )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $firstvalue | starting value of the linearly-spaced sequence to be generated |
| $lastvalue | final value of the linearly-spaced sequence to be generated |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The total number of matrix elements determines the step size ( ( lastvalue - firstvalue ) / (elements - 1) ).

First and last value cannot be identical.

*ColMajor* order means all rows are first filled in column zero, then in column one, and so forth.

## Related

*SetLinSpaced_Col*(), *SetLinSpaced_Diag*(), *SetLinSpaced_Row*(), *SetLinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
_Eigen_CleanUp()
```



SetLinSpaced_Diag

Function Reference

# _Eigen_SetLinSpaced_Diag

Fill all diagonal cells of a matrix with a succession of equally-spaced values

```
#include <Eigen4AutoIt.au3>
_Eigen_SetLinSpaced_Diag ( $matA, $firstvalue, $lastvalue )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $firstvalue | starting value of the linearly-spaced sequence to be generated |
| $lastvalue | final value of the linearly-spaced sequence to be generated |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The total number of diagonal elements determines the step size ( ( lastvalue - firstvalue ) / (elements - 1) ).

First and last value cannot be identical.

## Related

*SetLinSpaced_Col*(), *SetLinSpaced_ColMajor*(), *SetLinSpaced_Row*(), *SetLinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )
_Eigen_SetLinSpaced_Diag ( $matA, 1, 4 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```

# _Eigen_SetLinSpaced_Row

Fill all matrix cells in one row with a succession of equally-spaced values

```
#include <Eigen4AutoIt.au3>
_Eigen_SetLinSpaced_Row ( $matA, $rowindex, $firstvalue, $lastvalue )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |
| $firstvalue | starting value of the linearly-spaced sequence to be generated |
| $lastvalue | final value of the linearly-spaced sequence to be generated |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The total number of columns determines the step size ( ( lastvalue - firstvalue ) / (columns - 1) ).

First and last value cannot be identical.

## Related

*SetLinSpaced_Col*(), *SetLinSpaced_ColMajor*(), *SetLinSpaced_Diag*(), *SetLinSpaced_RowMajor*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )
$rowindex = 2
_Eigen_SetLinSpaced_Row ( $matA, $rowindex, 1, 4 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
_Eigen_CleanUp()
```



SetLinSpaced_RowMajor

Function Reference

# _Eigen_SetLinSpaced_RowMajor

Fill all matrix cells in RowMajor order with a succession of equally-spaced values

```
#include <Eigen4AutoIt.au3>
_Eigen_SetLinSpaced_RowMajor ( $matA, $firstvalue, $lastvalue )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $firstvalue | starting value of the linearly-spaced sequence to be generated |
| $lastvalue | final value of the linearly-spaced sequence to be generated |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The total number of matrix elements determines the step size ( ( lastvalue - firstvalue ) / (elements -1) ).

First and last value cannot be identical.

*RowMajor* order means all columns are first filled in row zero, then in row one, and so forth.

## Related

*SetLinSpaced_Col*(), *SetLinSpaced_ColMajor*(), *SetLinSpaced_Diag*(), *SetLinSpaced_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 6 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 18 )
```

```
_MatrixDisplay ( $matA, "LinSpaced RowMajor" )

_Eigen_CleanUp()
```





# _Eigen_SetOnes

Fill all matrix cells with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_SetOnes ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The matrix equivalent of a scalar value of 1 is an Identity matrix, not a matrix filled with ones.

## Related

*SetIdentity*(), *SetZero*(), *SetConstant*(), *SetOnes_Block*(), *SetOnes_Col*(), *SetOnes_Diag*(), *SetOnes_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetOnes ( $matA )

_MatrixDisplay ( $matA, "SetOnes" )

_Eigen_CleanUp()
```

SetOnes_Block

Function Reference

# _Eigen_SetOnes_Block

Fill all matrix cells in a block with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_SetOnes_Block ( $matA, $startRow, $startCol, $blockRows, $blockCols
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetZero_Block*(), *SetIdentity_Block*(), *SetConstant_Block*(), *SetOnes_Block*(), *SetOnes_Col*(), *SetOnes_Diag*(), *SetOnes_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 3, 4 )

$startRow = 1
$startCol = 1
$blockRows = 2
$blockCols = 2
_Eigen_SetOnes_Block ( $matA, $startRow, $startCol, $blockRows, $blockCols )
```

```
_MatrixDisplay ( $matA, "SetOnes_Block" )

_Eigen_CleanUp()
```



## SetOnes_Col

Function Reference

# _Eigen_SetOnes_Col

Fill all matrix cells in one column with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_SetOnes_Col ( $matA, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix act to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetOnes*(), *SetOnes_Block*(), *SetOnes_Diag*(), *SetOnes_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )

$colindex = 2
_Eigen_SetOnes_Col ( $matA, $colindex )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```

# _Eigen_SetOnes_Diag

Fill all diagonal cells of a matrix with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_SetOnes_Diag ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetOnes*(), *SetIdentity*(), *SetOnes_Block*(), *SetOnes_Col*(), *SetOnes_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )

_Eigen_SetOnes_Diag ( $matA )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```

# _Eigen_SetOnes_Row

Fill all matrix cells in a specific row with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_SetOnes_Row ( $matA, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetOnes*(), *SetOnes_Block*(), *SetOnes_Col*(), *SetOnes_Diag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )

$rowindex = 2
_Eigen_SetOnes_Row ( $matA, $rowindex )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```

# _Eigen_SetRandom

Fill all matrix cells with random values in the range 0 to ±1

```
#include <Eigen4AutoIt.au3>
_Eigen_SetRandom ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Parent distribution is flat.

## Related

*SetRandom_Block*(), *SetRandom_Col*(), *SetRandom_Diag*(), *SetRandom_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetRandom ( $matA )

_MatrixDisplay ( $matA, "SetRandom" )

_Eigen_CleanUp()
```





# _Eigen_SetRandom_Block

Fill all matrix cells in a block with random values in the range 0 to ±1

```
#include <Eigen4AutoIt.au3>
_Eigen_SetRandom_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Parent distribution is flat.

## Related

*SetRandom_Block*(), *SetRandom_Col*(), *SetRandom_Diag*(), *SetRandom_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 3, 4 )

$startRow = 1
$startCol = 1
$blockRows = 2
$blockCols = 2
_Eigen_SetRandom_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols )

_MatrixDisplay ( $matA, "SetRandom_Block" )

_Eigen_CleanUp()
```



### SetRandom_Col

Function Reference

# _Eigen_SetRandom_Col

Fill all matrix cells in a specific column with random values in the range 0 to ±1

```
#include <Eigen4AutoIt.au3>
_Eigen_SetRandom_Col ( $matA, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Parent distribution is flat.

## Related

*SetRandom*(), *SetRandom_Block*(), *SetRandom_Diag*(), *SetRandom_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )

$colindex = 2
_Eigen_SetRandom_Col ( $matA, $colindex )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```

# _Eigen_SetRandom_Diag

Fill all diagonal cells of a matrix with random values in the range 0 to ±1

```
#include <Eigen4AutoIt.au3>
_Eigen_SetRandom_Diag ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Parent distribution is flat.

## Related

*SetRandom*(), *SetRandom_Block*(), *SetRandom_Col*(), *SetRandom_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )

_Eigen_SetRandom_Diag ( $matA )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```



SetRandom_Row

Function Reference

# _Eigen_SetRandom_Row

Fill all matrix cells in one row with random values in the range 0 to ±1

```
#include <Eigen4AutoIt.au3>
_Eigen_SetRandom_Row ( $matA, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Parent distribution is flat.

## Related

*SetRandom*(), *SetRandom_Block*(), *SetRandom_Col*(), *SetRandom_Diag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Zero ( 4, 4 )

$rowindex = 2
_Eigen_SetRandom_Row ( $matA, $rowindex )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```





# _Eigen_SetZero

Fill all matrix cells with zero

```
#include <Eigen4AutoIt.au3>
_Eigen_SetZero ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

A "zero matrix" or "null matrix" is the matrix equivalent of a scalar value of zero.

## Related

*SetIdentity*(), *SetOnes*(), *SetConstant*(), *SetZero_Block*(), *SetZero_Col*(), *SetZero_Diag*(), *SetZero_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetZero ( $matA )

_MatrixDisplay ( $matA, "SetZero" )

_Eigen_CleanUp()
```



SetZero_Block

Function Reference

# _Eigen_SetZero_Block

Fill all matrix cells in a block with zero

```
#include <Eigen4AutoIt.au3>
_Eigen_SetZero_Block ( $matA, $startRow, $startCol, $blockRows, $blockCols
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

| | |
|---|---|
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetZero*(), *SetZero_Col*(), *SetZero_Diag*(), *SetZero_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 3, 4 )

$startRow = 1
$startCol = 1
$blockRows = 2
$blockCols = 2
_Eigen_SetZero_Block ( $matA, $startRow, $startCol, $blockRows, $blockCols )

_MatrixDisplay ( $matA, "SetZero_Block" )

_Eigen_CleanUp()
```





# _Eigen_SetZero_Col

Fill all matrix cells in one column with zero

```
#include <Eigen4AutoIt.au3>
_Eigen_SetZero_Col ( $matA, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetZero*(), *SetZero_Block*(), *SetZero_Diag*(), *SetZero_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )

$colindex = 2
_Eigen_SetZero_Col ( $matA, $colindex )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```



### SetZero_Diag

Function Reference

# _Eigen_SetZero_Diag

Fill all diagonal cells of a matrix with zero

```
#include <Eigen4AutoIt.au3>
_Eigen_SetZero_Diag ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*SetZero*(), *SetZero_Block*(), *SetZero_Col*(), *SetZero_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )

_Eigen_SetZero_Diag ( $matA )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```



### SetZero_Row

Function Reference

# _Eigen_SetZero_Row

Fill all matrix cells in one row with zero

```
#include <Eigen4AutoIt.au3>
_Eigen_SetZero_Row ( $matA, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

None.

## Related

*SetZero*(), *SetZero_Block*(), *SetZero_Col*(), *SetZero_Diag*()

## Example

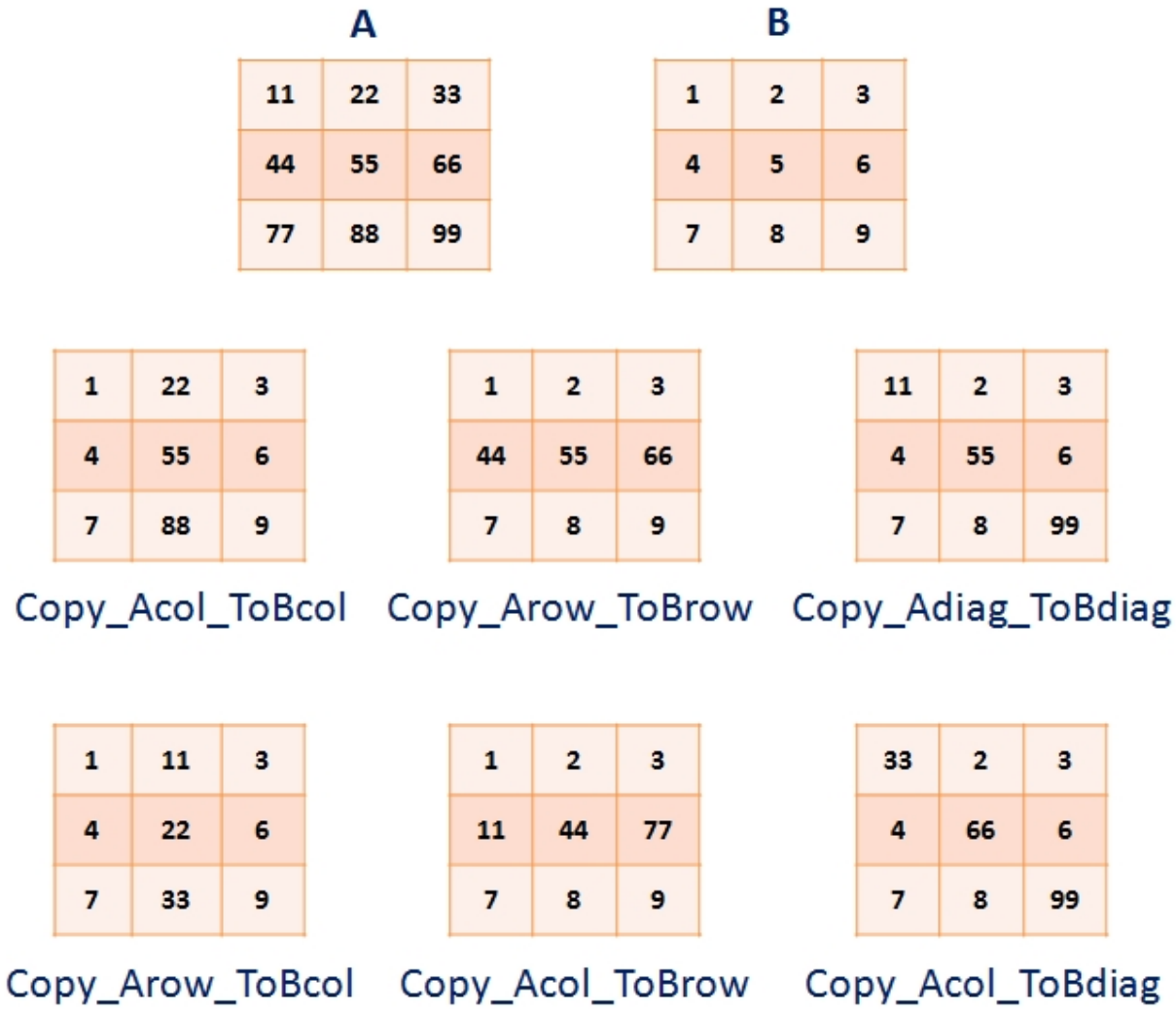
```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )

$rowindex = 2
_Eigen_SetZero_Row ( $matA, $rowindex )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```



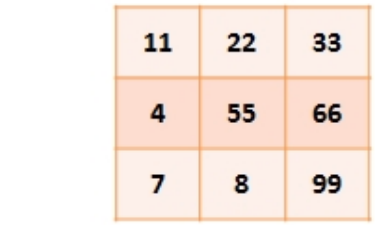
### Copy

# Copy

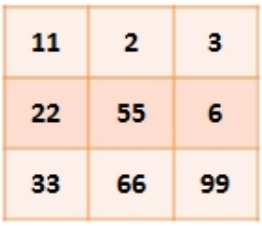
**Copy** functions duplicate data from one matrix (part) to another matrix (part). Both containers **need to exist** already and should have the appropriate dimension(s). Some examples, given the displayed initial contents of matrices **A** and **B**:



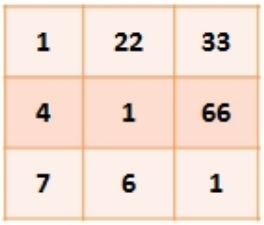

Some special cases involve the triiangular upper or lower parts of a matrix. These normally include the diagonal, which, however, is excluded if the infix "Strictly" is used, and filled with ones at the destination if the infix "Unit" is used. Some examples, given the same initial contents of matrices **A** and **B** as before:



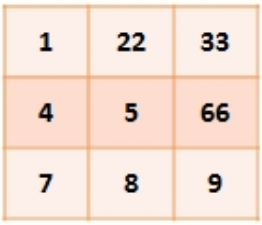

Copy_AUpper_ToBUpper




Copy_AUpper_ToBLower




Copy_AUnitUpper_ToBUpper




Copy_AStrictlyUpper_ToBUpper

Note that copying a triangular part involving a non-square source or destination matrix will fail unless the cell areas match.

Special Copy functions are available to transfer data between the real and/or imaginary parts of complex matrices. It is also possible to swap the real and imaginary parts of a single complex matrix.



## Copy_A_ToB

Function Reference

# _Eigen_Copy_A_ToB

Copy all data from matrix A to (existing) matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_A_ToB ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

Both matrices need to exist already; to create matrix **B** with **A**'s contents, call _Eigen_CloneMatrix() instead.

## Related

*Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetZero ( $matB )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_A_ToB ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_A_ToBimag

Copy all data from real matrix A to (existing) complex matrix B's imaginary part

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_A_ToBimag ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the **real** source matrix |
| $matB | matrix ID of the **complex** destination matrix |

## Return Value

| | |
|---|---|
| Success: | $matB |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and precision type. Matrix **A** needs to be real; matrix **B** needs to be complex.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Copy_Areal_ToB*(), *Copy_Aimag_ToB*(), *Copy_Areal_ToBreal*(), *Copy_Aimag_ToBimag*(), *Copy_Areal_ToBimag*(), *Copy_Aimag_ToBreal*(), *Copy_A_ToBreal*(), *Swap_Areal_Aimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")

$matA = _Eigen_CreateMatrix ( 4, 4, "float" )
_Eigen_SetConstant ( $matA, 123 )

_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_Copy_A_ToBimag ( $matA, $matB )

_MatrixDisplay_Imag ( $matB, "B Imag" )

_Eigen_CleanUp()
```





# _Eigen_Copy_A_ToBreal

Copy all data from real matrix A to (existing) complex matrix B's real part

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_A_ToBreal ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the **real** source matrix |
| $matB | matrix ID of the **complex** destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and precision type. Matrix **A** needs to be real; matrix **B** needs to be complex.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Copy_Areal_ToB*(), *Copy_Aimag_ToB*(), *Copy_Areal_ToBreal*(), *Copy_Aimag_ToBimag*(), *Copy_Areal_ToBimag*(), *Copy_Aimag_ToBreal*(), *Swap_Real_Imag*(), *Copy_A_ToBimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")

$matA = _Eigen_CreateMatrix ( 4, 4, "float" )
_Eigen_SetConstant ( $matA, 123 )

_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_Copy_A_ToBreal ( $matA, $matB )

_MatrixDisplay_Real ( $matB, "B real" )

_Eigen_CleanUp()
```



## Copy_Ablock_ToAblock

Function Reference

# _Eigen_Copy_Ablock_ToAblock

Copy all data from a block in matrix A to another block in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Ablock_ToAblock ( $matA, $startRow_src, $startCol_src,
$blockRows, $blockCols, $startRow_dst, $startCol_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow_src | row ID (base-0) of the topmost row of the block in source matrix A |
| $startCol_src | column ID (base-0) of the leftmost col of the block in source matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $startRow_dst | row ID (base-0) of the topmost row of the destination block in matrix A |
| $startCol_dst | column ID (base-0) of the leftmost col of the destination block in matrix A |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Source and destination block may (partially) overlap.

Block has to fit inside matrix at source and destination coordinates.

## Related

*Copy_Ablock_ToBblock()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "before copy" )

$startRow_src = 0
```

```
$startCol_src = 0
$blockRows = 2
$blockCols = 2
$startRow_dst = 2
$startCol_dst = 2

_Eigen_Copy_Ablock_ToAblock ( $matA, $startRow_src, $startCol_src, _
      $blockRows, $blockCols, $startRow_dst, $startCol_dst )

_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Ablock_ToBblock

Copy all data from a block in matrix A to a block in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Ablock_ToBblock ( $matA, $matB, $startRow_src, $startCol_src,
$blockRows, $blockCols, $startRow_dst, $startCol_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $startRow_src | row ID (base-0) of the topmost row of the block in source matrix A |
| $startCol_src | column ID (base-0) of the leftmost col of the block in source matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $startRow_dst | row ID (base-0) of the topmost row of the destination block in matrix B |
| $startCol_dst | column ID (base-0) of the leftmost col of the destination block in matrix B |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Block has to fit inside both matrices (may have different source and destination coordinates).

## Related

*Copy_Ablock_ToAblock()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )

$matB = _Eigen_CreateMatrix ( 5, 4 )
_Eigen_SetZero ( $matB )
_MatrixDisplay ( $matB, "before copy" )

$startRow_src = 0
$startCol_src = 0
$blockRows = 2
$blockCols = 2
$startRow_dst = 2
$startCol_dst = 2

_Eigen_Copy_Ablock_ToBblock ( $matA, $matB, $startRow_src, $startCol_src, _
      $blockRows, $blockCols, $startRow_dst, $startCol_dst )

_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Acol_ToBcol

Copy all data from a column in matrix A to a column in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, $colindex_src, $colindex_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |

| | |
|---|---|
| $colindex_dst | column ID (base-0) of the column in matrix B to copy the values to |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of rows in matrix **A** has to match the number of rows in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetZero ( $matB )
_MatrixDisplay ( $matB, "before copy" )

$colindex_src = 0
$colindex_dst = 2
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, $colindex_src, $colindex_dst )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Acol_ToBdiag

Copy all data from a column in matrix A to the diagonal of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Acol_ToBdiag ( $matA, $matB, $colindex_src )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of rows in matrix **A** has to match the number of diagonal cells in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetRandom ( $matA )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetZero ( $matB )
_MatrixDisplay ( $matB, "before copy" )

$colindex_src = 0
_Eigen_Copy_Acol_ToBdiag ( $matA, $matB, $colindex_src )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Acol_ToBrow

Copy all data from a column in matrix A to a row in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Acol_ToBrow ( $matA, $matB, $colindex_src, $rowindex_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row  in matrix B to copy the values to |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of rows in matrix **A** has to match the number of columns in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetRandom ( $matA )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetZero ( $matB )
_MatrixDisplay ( $matB, "before copy" )

$colindex_src = 0
$rowindex_dst = 2
_Eigen_Copy_Acol_ToBrow ( $matA, $matB, $colindex_src, $rowindex_dst )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



## Copy_Adiag_ToBcol

Function Reference

# _Eigen_Copy_Adiag_ToBcol

Copy all data from the diagonal of matrix A to a column of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Adiag_ToBcol ( $matA, $matB, $colindex_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $colindex_dst | column ID (base-0) of the column in matrix B to copy the values to |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of diagonal cells in matrix **A** has to match the number of rows in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

$colindex_dst = 2
_Eigen_Copy_Adiag_ToBrow ( $matA, $matB, $colindex_dst )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```

Copy_Adiag_ToBdiag

Function Reference

# _Eigen_Copy_Adiag_ToBdiag

Copy all data from the diagonal of matrix A to the diagonal of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Adiag_ToBdiag ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of diagonal cells in matrix **A** has to match the number of diagonal cells in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_Adiag_ToBdiag ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```

Copy_Adiag_ToBrow

Function Reference

# _Eigen_Copy_Adiag_ToBrow

Copy all data from the diagonal of matrix A to a row of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Adiag_ToBrow ( $matA, $matB, $rowindex_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $rowindex_dst | row ID (base-0) of the row  in matrix B to copy the values to |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of diagonal cells in matrix **A** has to match the number of columns in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

$rowindex_dst = 2
_Eigen_Copy_Adiag_ToBrow ( $matA, $matB, $rowindex_dst )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```

# _Eigen_Copy_Adiag_ToBvector

Copy all data from the diagonal of matrix A to vector B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Adiag_ToBvector ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix (vector) |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of diagonal cells in matrix **A** has to match the length of vector **B**.

Matrix **B** can be either a Colvector or a Rowvector.

## Related

*Copy_Avector_ToBdiag()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 1, 4 ) ; vector
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_Adiag_ToBvector ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```

# _Eigen_Copy_Aimag_ToB

Copy all data from complex matrix A's imaginary part to (existing) real matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Aimag_ToB ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the **complex** source matrix |
| $matB | matrix ID of the **real** destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and precision type. Matrix **A** needs to be complex; matrix **B** needs to be real.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Copy_Areal_ToB*(), *Swap_Real_Imag*(), *Copy_Areal_ToBreal*(), *Copy_Aimag_ToBimag*(), *Copy_Areal_ToBimag*(), *Copy_Aimag_ToBreal*(), *Copy_A_ToBreal*(), *Copy_A_ToBimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetConstant_Real ( $matA, 123 )
_Eigen_SetConstant_Imag ( $matA, 456 )

_MatrixDisplay_Real ( $matA, "A, real" )
_MatrixDisplay_Real ( $matA, "A, imag" )
```

```
$matB = _Eigen_CreateMatrix ( 4, 4, "float" )
_Eigen_Copy_Aimag_ToB ( $matA, $matB )

_MatrixDisplay ( $matB, "matrix B" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Aimag_ToBimag

# add EXAMPLE

Copy all data from complex matrix A's imaginary part to (existing) complex matrix B's imaginary part

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Aimag_ToBimag ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Complex matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and matrix type.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Copy_Areal_ToB*(), *Copy_Aimag_ToB*(), *Copy_Areal_ToBreal*(), *Swap_Real_Imag*(), *Copy_Areal_ToBimag*(), *Copy_Aimag_ToBreal*(), *Copy_A_ToBreal*(), *Copy_A_ToBimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetConstant_Real ( $matA, 123 )
_Eigen_SetConstant_Imag ( $matA, 456 )

_MatrixDisplay_Real ( $matA, "A, real" )
_MatrixDisplay_Imag ( $matA, "A, imag" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_Copy_Aimag_ToBimag ( $matA, $matB )

_MatrixDisplay_Imag ( $matB, "B imag" )

_Eigen_CleanUp()
```



### Copy_Aimag_ToBreal

Function Reference

# _Eigen_Copy_Aimag_ToBreal

Copy all data from complex matrix A's imaginary part to (existing) complex matrix B's real part

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Aimag_ToBreal ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the **complex** source matrix |
| $matB | matrix ID of the **complex** destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Complex matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and matrix type.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Copy_Areal_ToB*(), *Copy_Aimag_ToB*(), *Copy_Areal_ToBreal*(), *Copy_Aimag_ToBimag*(), *Copy_Areal_ToBimag*(), *Swap_Real_Imag*(), *Copy_A_ToBreal*(), *Copy_A_ToBimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetConstant_Real ( $matA, 123 )
_Eigen_SetConstant_Imag ( $matA, 456 )

_MatrixDisplay_Real ( $matA, "A, real" )
_MatrixDisplay_Imag ( $matA, "A, imag" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_Copy_Aimag_ToBreal ( $matA, $matB )

_MatrixDisplay_Real ( $matB, "B real" )

_Eigen_CleanUp()
```



### Copy_ALower_ToBLower

Function Reference

# _Eigen_Copy_ALower_ToBLower

Copy all data from the lower triangular of matrix A to the lower triangular of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_ALower_ToBLower ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

## Related

*Copy_AUpper_ToBUpper*(), *Copy_AUpper_ToBLower*(), *Copy_ALower_ToBUpper*(), *Copy_AUnitUpper_ToBUpper*(), *Copy_AUnitLower_ToBLower*(), *Copy_AStrictlyUpper_ToBUpper*(), *Copy_AStrictlyLower_ToBLower*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_ALower_ToBLower ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_ALower_ToBUpper

Function Reference

# _Eigen_Copy_ALower_ToBUpper

Copy all data from the lower triangular of matrix A to the upper triangular of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_ALower_ToBUpper ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

| | |
|---|---|
| Success: | $matB |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

## Related

*Copy_AUpper_ToBUpper*(), *Copy_AUpper_ToBLower*(), *Copy_ALower_ToBLower*(), *Copy_AUnitUpper_ToBUpper*(), *Copy_AUnitLower_ToBLower*(), *Copy_AStrictlyUpper_ToBUpper*(), *Copy_AStrictlyLower_ToBLower*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_ALower_ToBUpper ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_Arow_ToBcol

Function Reference

# _Eigen_Copy_Arow_ToBcol

Copy all data from a row of matrix A to a column of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Arow_ToBcol ( $matA, $matB, $rowindex_src, $colindex_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column  in matrix B to copy the values to |

## Return Value

| | |
|---|---|
| Success: | $matB |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

$rowindex_src = 0
$colindex_dst = 2
_Eigen_Copy_Arow_ToBcol ( $matA, $matB, $rowindex_src, $colindex_dst )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Arow_ToBdiag

Copy all data from a row in matrix A to the diagonal of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Arow_ToBdiag ( $matA, $matB, $rowindex_src )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |

## Return Value

| | |
|---|---|
| Success: | $matB |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

$rowindex_src = 0
_Eigen_Copy_Arow_ToBdiag ( $matA, $matB, $rowindex_src )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Arow_ToBrow

Copy all data from a row of matrix A to a row of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Arow_ToBrow ( $matA, $matB, $rowindex_src, $rowindex_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

| | |
|---|---|
| $matB | matrix ID of the destination matrix |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix B to copy the values to |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of columns in matrix **B**.

## Related

*Copy_A_ToB*(), *Copy_Acol_ToBcol*(), *Copy_Arow_ToBrow*(), *Copy_Acol_ToBrow*(), *Copy_Arow_ToBcol*(), *Copy_Adiag_ToBdiag*(), *Copy_Acol_ToBdiag*(), *Copy_Arow_ToBdiag*(), *Copy_Adiag_ToBrow*(), *Copy_Adiag_ToBcol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

$rowindex_src = 0
$rowindex_dst = 2
_Eigen_Copy_Arow_Torow ( $matA, $matB, $rowindex_src, $rowindex_dst )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```





# _Eigen_Copy_ArrayData_ToMatrix

Copy all array data to a same-sized, existing matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_ArrayData_ToMatrix ( $matA, ByRef $array[, $matrixType =
$EIGEN_MATRIXTYPE[, $convertScientificNotation = False[, $imag = False ]]]
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $array | source array |
| $matrixType | **[optional]** "int", "float", "double", "complexfloat", or "complexdouble" |
| $convertScientificNotation | **[optional]** **True**: convert values to scientific notation strings; **False**: copy as-is (faster) |
| $imag | **[optional]** **True**: access imaginary part of a complex matrix; **False**: access real part |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

By default, the array is assumed to contain numeric values. If, however, scientific notation strings may (also) be present, these can be detected and converted (slower). Any remaining text string is stored in a matrix as the value zero.

Complex variants with suffix _Real/_Imag are available; calling the main function on a complex matrix accesses the real part, unless $imag = True (default: False).

## Related

*Copy_MatrixData_ToArray()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

Local $arrayA[3][3] = [[1,2,3], [4,5,6], [7,8,9]]
_ArrayDisplay ( $arrayA, "array A" )

$matA = _Eigen_CreateMatrix ( 3, 3 )      ; same dimensions as array!
_Eigen_Copy_ArrayData_ToMatrix ( $matA, $arrayA )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_CleanUp()
```



## Copy_Areal_ToB

Function Reference

# _Eigen_Copy_Areal_ToB

Copy all data from complex matrix A's real part to (existing) real matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Areal_ToB ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the **complex** source matrix |
| $matB | matrix ID of the **real** destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and precision type. Matrix **A** needs to be complex; matrix **B** needs to be real.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Swap_Real_Imag*(), *Copy_Aimag_ToB*(), *Copy_Areal_ToBreal*(), *Copy_Aimag_ToBimag*(), *Copy_Areal_ToBimag*(), *Copy_Aimag_ToBreal*(), *Copy_A_ToBreal*(), *Copy_A_ToBimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetConstant_Real ( $matA, 123 )
_Eigen_SetConstant_Imag ( $matA, 456 )

_MatrixDisplay_Real ( $matA, "A, real" )
_MatrixDisplay_Real ( $matA, "A, imag" )

$matB = _Eigen_CreateMatrix ( 4, 4, "float" )
_Eigen_Copy_Areal_ToB ( $matA, $matB )

_MatrixDisplay ( $matB, "B" )

_Eigen_CleanUp()
```

# _Eigen_Copy_Areal_ToBimag

Copy all data from complex matrix A's real part to (existing) complex matrix B's imaginary part

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Areal_ToBimag ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the **complex** source matrix |
| $matB | matrix ID of the **complex** destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Complex matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and matrix type.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Copy_Areal_ToB*(), *Copy_Aimag_ToB*(), *Copy_Areal_ToBreal*(), *Copy_Aimag_ToBimag*(), *Swap_Real_Imag*(), *Copy_Aimag_ToBreal*(), *Copy_A_ToBreal*(), *Copy_A_ToBimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetConstant_Real ( $matA, 123 )
_Eigen_SetConstant_Imag ( $matA, 456 )

_MatrixDisplay_Real ( $matA, "A, real" )
_MatrixDisplay_Imag ( $matA, "A, imag" )
```

```
$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_Copy_Areal_ToBimag ( $matA, $matB )

_MatrixDisplay_Imag ( $matB, "B imag" )

_Eigen_CleanUp()
```





# _Eigen_Copy_Areal_ToBreal

Copy all data from complex matrix A's real part to (existing) complex matrix B's real part

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Areal_ToBreal ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the **complex** source matrix |
| $matB | matrix ID of the **complex** destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Complex matrices **A** and **B** have to have the **same** dimensions (rows *and* cols) and matrix type.

Both matrices need to exist already.

## Related

*Copy_A_ToB*(), *Copy_Areal_ToB*(), *Copy_Aimag_ToB*(), *Swap_Real_Imag*(), *Copy_Aimag_ToBimag*(), *Copy_Areal_ToBimag*(), *Copy_Aimag_ToBreal*(), *Copy_A_ToBreal*(), *Copy_A_ToBimag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("complexfloat")
```

```
$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetConstant_Real ( $matA, 123 )
_Eigen_SetConstant_Imag ( $matA, 456 )

_MatrixDisplay_Real ( $matA, "A, real" )
_MatrixDisplay_Imag ( $matA, "A, imag" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_Copy_Areal_ToBreal ( $matA, $matB )

_MatrixDisplay_Real ( $matB, "B real" )

_Eigen_CleanUp()
```





# _Eigen_Copy_AStrictlyLower_ToBLower

Copy all data below the diagonal of matrix A to the same area in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_AStrictlyLower_ToB ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

"Strictly" means that diagonal data are not copied across.

## Related

*Copy_AUpper_ToBUpper*(), *Copy_AUpper_ToBLower*(), *Copy_ALower_ToBUpper*(), *Copy_AUnitUpper_ToBUpper*(), *Copy_AUnitLower_ToBLower*(), *Copy_AStrictlyUpper_ToBUpper*(), *Copy_ALower_ToBLower*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_AStrictlyLower_ToBLower ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_AStrictlyUpper_ToBUpper

Function Reference

# _Eigen_Copy_AStrictlyUpper_ToBUpper

Copy all data above the diagonal of matrix A to the same area in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_AStrictlyUpper_ToBUpper ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

"Strictly" means that diagonal data are not copied across.

## Related

*Copy_AUpper_ToBUpper*(), *Copy_AUpper_ToBLower*(), *Copy_ALower_ToBUpper*(), *Copy_AUnitUpper_ToBUpper*(), *Copy_AUnitLower_ToBLower*(), *Copy_ALower_ToBLower*(), *Copy_AStrictlyLower_ToBLower*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_AStrictlyUpper_ToBUpper ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_AUnitLower_ToBLower

Function Reference

# _Eigen_Copy_AUnitLower_ToBLower

Copy all data below the diagonal in matrix A to the same area in matrix B, and fill the latter's diagonal with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_AUnitLower_ToBLower ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Square matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

"Unit" means that the destination's diagonal is filled with ones.

## Related

*Copy_AUpper_ToBUpper*(), *Copy_AUpper_ToBLower*(), *Copy_ALower_ToBUpper*(), *Copy_AUnitUpper_ToBUpper*(), *Copy_ALower_ToBLower*(), *Copy_AStrictlyUpper_ToBUpper*(), *Copy_AStrictlyLower_ToBLower*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_AUnitLower_ToBLower ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_AUnitUpper_ToBUpper

Function Reference

# _Eigen_Copy_AUnitUpper_ToBUpper

Copy all data above the diagonal in matrix A to the same area in matrix B, and fills the latter's diagonal with ones

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_AUnitUpper_ToBUpper ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

"Unit" means that the destination's diagonal is filled with ones.

## Related

*Copy_AUpper_ToBUpper*(), *Copy_AUpper_ToBLower*(), *Copy_ALower_ToBUpper*(), *Copy_ALower_ToBLower*(), *Copy_AUnitLower_ToBLower*(), *Copy_AStrictlyUpper_ToBUpper*(), *Copy_AStrictlyLower_ToBLower*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_AUnitUpper_ToBUpper ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_AUpper_ToBLower

Function Reference

# _Eigen_Copy_AUpper_ToBLower

Copy all data from the upper triangular of matrix A to the lower triangular of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_AUpper_ToBLower ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Square matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

## Related

*Copy_AUpper_ToBUpper*(), *Copy_ALower_ToBUpper*(), *Copy_ALower_ToBLower*(), *Copy_AUnitLower_ToBLower*(), *Copy_AStrictlyUpper_ToBUpper*(), *Copy_AStrictlyLower_ToBLower*(), *Copy_AUnitUpper_ToBUpper*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_AUpper_ToBLower ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_AUpper_ToBUpper

Function Reference

# _Eigen_Copy_AUpper_ToBUpper

Copy all data from the upper triangular of matrix A to the upper triangular of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_AUpper_ToBUpper ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

## Related

*Copy_AUpper_ToBLower*(), *Copy_ALower_ToBUpper*(), *Copy_ALower_ToBLower*(), *Copy_AUnitLower_ToBLower*(), *Copy_AStrictlyUpper_ToBUpper*(), *Copy_AStrictlyLower_ToBLower*(), *Copy_AUnitUpper_ToBUpper*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_AUpper_ToBUpper ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_Avector_ToBdiag

Function Reference

# _Eigen_Copy_Avector_ToBdiag

Copy all data from vector A to the diagonal of matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_Avector_ToBdiag ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix (vector) |
| $matB | matrix ID of the destination matrix |

## Return Value

Success: $matB

Failure: False, and sets the @error flag to non-zero.

## Remarks

The length of vector **A** has to match the number of diagonal cells in matrix **B**.

Matrix **A** can be either a Colvector or a Rowvector.

## Related

*Copy_Adiag_ToBvector()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 1, 4 )     ; vector
$matB = _Eigen_CreateMatrix_Zero ( 4, 4 )
_MatrixDisplay ( $matB, "before copy" )

_Eigen_Copy_Avector_ToBdiag ( $matA, $matB )
_MatrixDisplay ( $matB, "after copy" )

_Eigen_CleanUp()
```



### Copy_MatrixData_ToArray

Function Reference

# _Eigen_Copy_MatrixData_ToArray

Copy all matrix data to a same-sized, existing array

```
#include <Eigen4AutoIt.au3>
_Eigen_Copy_MatrixData_ToArray ( $matA, ByRef $array[,
$convertScientificNotation = False[, $imag = False )] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $array | destination array |
| $convertScientificNotation | **[optional]** **True**: convert values to scientific notation strings; **False**: copy as-is (faster) |
| $imag | **[optional]** **True**: access imaginary part of a complex matrix; **False**: access real part |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

By default, the array is assumed to store numeric values. If, however, scientific notation strings are desired, these can be produced if the appropriate flag is set (slower).

Complex variants with suffix _Real/_Imag are available; calling the main function on a complex matrix accesses the real part, unless $imag = True (default: False).

## Related

*Copy_ArrayData_ToMatrix()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )

Local $arrayA [3][4]    ; needs to be already declared with same dimensions as
matrix
_Eigen_Copy_MatrixData_ToArray ( $matA, $arrayA )
_ArrayDisplay ( $arrayA, "array A" )

_Eigen_CleanUp()
```



### Swap

# Swap

**Swap** functions simply exchange the cell contents of two designated matrix parts, which have to have the same number of cells. The swap can be performed either inside a single matrix, or between two matrices. If acting on the parts of a single matrix and two dimensions are involved, the matrix has to be square. Note that reverse counterparts for matrix pair functions are not explicitly provided, as their objective can be achieved by simply parsing the two matrix IDs in reverse order. For example, swapping *Acol* with *Brow* is provided explicitly, but swapping *Arow* with *Bcol* is not, as one can simply call the former function with the matrix ID parsing order exchanged: Swap_Acol_Brow ( $matB, $matA, $colindex, $rowindex )), in which the column index then refers to matrix **B** (the first parsed matrix ID), and the row index refers to matrix **A** (the second parsed matrix ID).

- Swap_Ablock_Ablock
- Swap_Ablock_Bblock
- Swap_Acol_Acol
- Swap_Acol_Adiag
- Swap_Acol_Arow
- Swap_Acol_Bcol
- Swap_Acol_Bdiag

- Swap_Acol_Brow
- Swap_Adiag_Bdiag
- Swap_ALower_BLower
- Swap_Areal_Aimag
- Swap_Areal_Bimag
- Swap_Arow_Adiag
- Swap_Arow_Arow
- Swap_Arow_Bdiag
- Swap_Arow_Brow
- Swap_AStrictlyLower_BStrictlyLower
- Swap_AStrictlyUpper_BStrictlyLower
- Swap_AStrictlyUpper_BStrictlyUpper
- Swap_AUpper_ALower
- Swap_AUpper_BLower
- Swap_AUpper_BUpper





# _Eigen_Swap_Ablock_Ablock

Exchange the contents of two specified blocks of a single matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Ablock_Ablock ( $matA, $startRow_src, $startCol_src,
$blockRows, $blockCols, $startRow_dst, $startCol_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow_src | row ID (base-0) of the topmost row of the block in source matrix A |
| $startCol_src | column ID (base-0) of the leftmost col of the block in source matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $startRow_dst | row ID (base-0) of the topmost row of the destination block in matrix A |
| $startCol_dst | column ID (base-0) of the leftmost col of the destination block in matrix A |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

None.

## Related

*Swap_Ablock_Bblock()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "before swap" )

$startRow_src = 0
$startCol_src = 0
$blockRows = 2
$blockCols = 2
$startRow_dst = 2
$startCol_dst = 2

_Eigen_Swap_Ablock_Ablock ( $matA, $startRow_src, $startCol_src, _
     $blockRows, $blockCols, $startRow_dst, $startCol_dst )

_MatrixDisplay ( $matA, "after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_Ablock_Bblock

Exchange the contents of a specified block in one matrix with an equal-sized block in another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Ablock_Ablock ( $matA, $matB, $startRow_src, $startCol_src,
$blockRows, $blockCols, $startRow_dst, $startCol_dst )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |
| $startRow_src | row ID (base-0) of the topmost row of the block in source matrix A |
| $startCol_src | column ID (base-0) of the leftmost col of the block in source matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $startRow_dst | row ID (base-0) of the topmost row of the destination block in matrix B |
| $startCol_dst | column ID (base-0) of the leftmost col of the destination block in matrix B |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Ablock_Ablock()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

$startRow_src = 0
$startCol_src = 0
$blockRows = 2
$blockCols = 2
$startRow_dst = 2
$startCol_dst = 2

_Eigen_Swap_Ablock_Bblock ( $matA, $matB, $startRow_src, $startCol_src, _
     $blockRows, $blockCols, $startRow_dst, $startCol_dst )
```

```
_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_Acol_Acol

Exchange the contents of two specified columns of a single matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Acol_Acol ( $matA, $colindex1, $colindex2 )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex1 | column ID (base-0) of the first column in matrix A to act upon |
| $colindex2 | column ID (base-0) of the second column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Acol_Adiag*(), *Swap_Acol_Arow*(), *Swap_Acol_Bcol*(), *Swap_Acol_Bdiag*(), *Swap_Acol_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "before swap" )

_Eigen_Swap_Acol_Acol ( $matA, 1, 2 )
_MatrixDisplay ( $matA, "after swap" )
```

```
_Eigen_CleanUp()
```



## Swap_Acol_Adiag

Function Reference

# _Eigen_Swap_Acol_Adiag

Exchange the contents of a specified column with the contents of the diagonal

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Acol_Adiag ( $matA, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Acol_Acol*(), *Swap_Acol_Arow*(), *Swap_Acol_Bcol*(), *Swap_Acol_Bdiag*(), *Swap_Acol_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "before swap" )

_Eigen_Swap_Acol_Adiag ( $matA, 1 )
_MatrixDisplay ( $matA, "after swap" )

_Eigen_CleanUp()
```

## Swap_Acol_Arow

Function Reference

# _Eigen_Swap_Acol_Arow

Exchange the contents of a specified column and row of a single, square matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Acol_Arow ( $matA, $colindex, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrix **A** has to be **square**.

## Related

*Swap_Acol_Acol*(), *Swap_Acol_Adiag*(), *Swap_Acol_Bcol*(), *Swap_Acol_Bdiag*(), *Swap_Acol_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "before swap" )

_Eigen_Swap_Acol_Arow ( $matA, 1, 2 )
_MatrixDisplay ( $matA, "after swap" )

_Eigen_CleanUp()
```



## Swap_Acol_Bcol

Function Reference

# _Eigen_Swap_Acol_Bcol

Exchange the contents of a specified column in one matrix with a specified, equal-sized column in another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Acol_Bcol ( $matA, $matB, $colindex1, $colindex2 )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |
| $colindex1 | column ID (base-0) of the first column in matrix A to act upon |
| $colindex2 | column ID (base-0) of the second column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Acol_Acol*(), *Swap_Acol_Adiag*(), *Swap_Acol_Arow*(), *Swap_Acol_Bdiag*(), *Swap_Acol_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_Acol_Bcol ( $matA, 1, 2 )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```

# _Eigen_Swap_Acol_Bdiag

Exchange the contents of a specified column with the contents of another matrix's diagonal

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Acol_Bdiag ( $matA, $natB, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Acol_Acol*(), *Swap_Acol_Adiag*(), *Swap_Acol_Arow*(), *Swap_Acol_Bcol*(), *Swap_Acol_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_Acol_Bdiag ( $matA, 1 )
```

```
_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_Acol_Brow

Exchange the contents of a specified column in one matrix with a specified, equal-sized row in another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Acol_Brow ( $matA, $matB, $colindex, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |
| $colindex | column ID (base-0) of the column in matrix A to act upon |
| $rowindex | row ID (base-0) of the row in matrix B to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Acol_Acol*(), *Swap_Acol_Adiag*(), *Swap_Acol_Arow*(), *Swap_Acol_Bcol*(), *Swap_Acol_Bdiag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )
```

```
$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_Acol_Brow ( $matA, 1, 2 )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```



## Swap_Adiag_Bdiag

Swap_Adiag_BdiagFunction Reference

# _Eigen_Swap_Adiag_Bdiag

Exchange the contents of the diagonals between two matrices

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Adiag_Bdiag ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Acol_Adiag*(), *Swap_Acol_Bdiag*(), *Swap_Arow_Adiag*(), *Swap_Arow_Bdiag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )
```

```
$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_Adiag_Bdiag ( $matA, $matB )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_ALower_BLower

Exchange the contents of the lower triangular parts between two matrices

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_ALower_BLower ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Includes the diagonal.

## Related

*Swap_AUpper_ALower*(), *Swap_AUpper_BLower*(), *Swap_AUpper_BUpper*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
```

```
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_AUpper_BLower ( $matA, $matB )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_Areal_Aimag

Exchange the real and imaginary parts of a single complex  matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Areal_Aimag ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

This function accepts only complex inputs.

## Related

*Swap_Areal_Bimag()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp( "complexfloat" )

$matA = _Eigen_CreateMatrix ( 3, 4 )
```

```
_Eigen_SetZero_Real ( $matA )
_Eigen_SetOnes_Imag ( $matA )

_MatrixDisplay_Real ( $matA, "Real, before swap" )
_MatrixDisplay_Imag ( $matA, "Imag, before swap" )

_Eigen_Swap_Areal_Aimag ( $matA, $matB )

_MatrixDisplay_Real ( $matA, "Real, after swap" )
_MatrixDisplay_Imag ( $matA, "Imag, after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_Areal_Bimag

Exchange the real part of one complex matrix with imaginary part of another complex matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Areal_Bimag ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

This function accepts only complex inputs.

## Related

*Swap_Areal_Aimag()*

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp( "complexfloat" )

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetZero_Real ( $matA )
_Eigen_SetOnes_Imag ( $matA )

_MatrixDisplay_Real ( $matA, "A Real, before swap" )
_MatrixDisplay_Imag ( $matA, "A Imag, before swap" )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetOnes_Real ( $matB )
_Eigen_SetZero_Imag ( $matB )

_MatrixDisplay_Real ( $matB, "B Real, before swap" )
_MatrixDisplay_Imag ( $matB, "B Imag, before swap" )

_Eigen_Swap_Areal_Bimag ( $matA, $matB )

_MatrixDisplay_Real ( $matA, "A Real, after swap" )
_MatrixDisplay_Imag ( $matA, "A Imag, after swap" )

_MatrixDisplay_Real ( $matB, "B Real, after swap" )
_MatrixDisplay_Imag ( $matB, "B Imag, after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_Arow_Adiag

Exchange the contents of a specified row with the contents of the diagonal

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Arow_Adiag ( $matA, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the first column in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Arow_Arow*(), *Swap_Arow_Bdiag*(), *Swap_Arow_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "before swap" )

_Eigen_Swap_Arow_Adiag ( $matA, 1 )
_MatrixDisplay ( $matA, "after swap" )

_Eigen_CleanUp()
```



### Swap_Arow_Arow

Function Reference

# _Eigen_Swap_Arow_Arow

Exchange the contents of two specified rows of a single matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Arow_Arow ( $matA, $rowindex1, $rowindex2 )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex1 | row ID (base-0) of the first row in matrix A to act upon |
| $rowindex2 | row ID (base-0) of the second row in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Arow_Adiag*(), *Swap_Arow_Bdiag*(), *Swap_Arow_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "before swap" )

_Eigen_Swap_Arow_Arow ( $matA, 1, 2 )
_MatrixDisplay ( $matA, "after swap" )

_Eigen_CleanUp()
```



### Swap_Arow_Bdiag

Function Reference

# _Eigen_Swap_Arow_Bdiag

Exchange the contents of a specified column with the contents of another matrix's diagonal

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Arow_Bdiag ( $matA, $natB, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_Arow_Adiag*(), *Swap_Arow_Arow*(), *Swap_Arow_Brow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_Arow_Bdiag ( $matA, 1 )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_Arow_Brow

Exchange the contents of a specified row in one matrix with a specified, equal-sized row in another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Arow_Brow ( $matA, $matB, $rowindex1, $rowindex2 )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |
| $rowindex1 | row ID (base-0) of the row in matrix A to act upon |
| $rowindex2 | row ID (base-0) of the row in matrix B to act upon |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

None.

## Related

*Swap_Arow_Adiag*(), *Swap_Arow_Arow*(), *Swap_Arow_Bdiag*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_Arow_Brow ( $matA, 1, 2 )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_AStrictlyLower_BStrictlyLower

Exchange the contents of the strictly-lower triangular part of one matrix with the equal-sized, strictly-lower triangular part of another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_AStrictlyLower_BStrictlyLower ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Excludes the diagonal.

## Related

*Swap_AStrictlyUpper_BStrictlyLower*(), *Swap_AStrictlyUpper_BStrictlyUpper*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_AStrictlyLower_BStrictlyLower ( $matA, $matB )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```





# _Eigen_Swap_AStrictlyUpper_BStrictlyLower

Exchange the contents of the strictly-upper triangular part of one matrix with the equal-

sized, strictly lower triangular part of another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_AStrictlyUpper_BStrictlyLower ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Excludes the diagonal.

## Related

*Swap_AStrictlyLower_BStrictlyLower*(), *Swap_AStrictlyUpper_BStrictlyUpper*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_AStrictlyUpper_BStrictlyLower ( $matA, $matB )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```



## Swap_AStrictlyUpper_BStrictlyUpper

Function Reference

# _Eigen_Swap_AStrictlyUpper_BStrictlyUpper

Exchange the contents of the strictly-upper triangular part of one matrix with the equal-sized, strictly-upper triangular part of another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_AStrictlyUpper_BStrictlyUpper ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Excludes the diagonal.

## Related

*Swap_AStrictlyLower_BStrictlyLower()*, *Swap_AStrictlyUpper_BStrictlyLower()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_AStrictlyUpper_BStrictlyUpper ( $matA, $matB )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```

Swap_AUpper_ALower

Function Reference

# _Eigen_Swap_AUpper_ALower

Exchange the contents of the upper and lower triangular parts of a single, square matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_Acol_Acol ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

None.

## Related

*Swap_ALower_BLower*(), *Swap_AUpper_BLower*(), *Swap_AUpper_BUpper*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "before swap" )

_Eigen_Swap_AUpper_ALower ( $matA )
_MatrixDisplay ( $matA, "after swap" )

_Eigen_CleanUp()
```



Swap_AUpper_BLower

Function Reference

# _Eigen_Swap_AUpper_BLower

Exchange the contents of the upper triangular part of one matrix with the equal-sized, lower triangular part of another matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_AUpper_BLower ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Includes the diagonal.

## Related

*Swap_ALower_BLower*(), *Swap_AUpper_ALower*(), *Swap_AUpper_BUpper*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_AUpper_BLower ( $matA, $matB )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```



Swap_AUpper_BUpper

# _Eigen_Swap_AUpper_BUpper

Exchange the contents of the upper triangular parts between two matrices

```
#include <Eigen4AutoIt.au3>
_Eigen_Swap_AUpper_BUpper ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first matrix to act upon |
| $matB | matrix ID of the second matrix to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Includes the diagonal.

## Related

*Swap_ALower_BLower*(), *Swap_AUpper_ALower*(), *Swap_AUpper_BLower*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 1, 16 )
_MatrixDisplay ( $matA, "A before swap" )

$matB = _Eigen_CreateMatrix_LinSpaced_RowMajor ( 4, 4, 101, 116 )
_MatrixDisplay ( $matB, "B before swap" )

_Eigen_Swap_AUpper_BUpper ( $matA, $matB )

_MatrixDisplay ( $matA, "A after swap" )
_MatrixDisplay ( $matB, "B after swap" )

_Eigen_CleanUp()
```

### Cellwise Operators

# Cellwise Operators

"Cwise" is short for Cell-wise (or Coefficient-wise), meaning that Cwise operators act on **each** matrix cell **individually**. Three different basic types exist:

**_Eigen_CwiseUnaryOp** applies a specified operator directly to each cell, e.g., **abs,sqrt,sin,cube...**
**_Eigen_CwiseScalarOp** applies {operator: $scalar} to each cell: **+,-,/,*,min,max,pow** only
**_Eigen_CwiseBinaryOp** applies {operator: [$matB cell]} to **A** cell: **+,-,/,*,min,max**

The operator itself is parsed as a string, like so:

**_Eigen_CwiseUnaryOp (** $matA, "square" **)** to fill each cell with its value squared;
**_Eigen_CwiseScalarOp (** $matA, "+", **3 )** to add scalar *3* to each cell's original value
**_Eigen_CwiseBinaryOp (** $matA, "/", $matB **)** to divide each cell in matrix **A** with the corresponding cell's value in matrix **B**. Here, matrices **A** and **B** *have* to have the **same dimensions**!

Note that the matrix type may impose restrictions on the available operators, notably in the case of type integer, in which the following real-returning unary operators are unavailable: acos, asin, cos, exp, inverse, log, sin, sqrt, tan.

The result of the operation can be returned either in a separate results matrix (the default), or in-place (when suffix **_InPlace** is added to the function name).

Just like **Set** and **Copy** functions, **Cwise Op**s also exist in many subset-specific variations designating source and destination. Some examples:

**_Eigen_CwiseUnaryOp_Row** act on each cell in a specific row
**_Eigen_CwiseBinaryOp_Block** act on each cell inside a block
**_Eigen_CwiseScalarOp_ColCol** operate on cells in one col, store in another col
**_Eigen_CwiseUnaryOp_ColRow** operate on cells in one col, store in another **row**

Functions **_Eigen_CwiseBinaryOp_Rowwise/Colwise(**$matA, $operator, **$vecB)** are special. Most BinaryOp functions take two equal-sized matrices, but here a vector is supplied, and the operator is applied repeatedly to each row or column of the target matrix. In this case, any vector type (row/col) will do; the dll wrapper will automatically map it in the appropriate dimension. If, instead, you wish to use a single row or column from a source matrix **B** to apply to all rows or cols, you can use commands such as: **_Eigen_CwiseBinaryOp_ColwiseCol**($matA, $colindex_src, $operator, **$matB**), where matrices **A** and **B** have to have the same **alternate** dimension (so in the ColwiseCol example, they should have an equal number of rows).

**Cwise** Unary, Scalar, and Binary operators can also be applied **conditionally**, using functions of the format:

**_Eigen_ConditUnaryOp** ( $matA, $conditOperator, $conditScalar, $unaryOperator )
**_Eigen_ConditScalarOp** ( $matA, $conditOperator, $conditScalar, $scalarOperator, $scalar )
**_Eigen_ConditBinaryOp** ( $matA, $conditOperator, $conditScalar, $binaryOperator, $matB )

Conditional Operators compare a separately supplied scalar (parameter $conditscalar) with the cell content (prior to performing the operation), and *only* performs the unary/scalar/binary operation if the

comparison yields True. The comparison itself can test equality, inequality, larger, smaller, larger-or-equal, and smaller-or-equal.

Conditional operators can also yield matrix specs, notably:

**_Eigen_ConditAll** returns True if all matrix cells satisfy the condition
**_Eigen_ConditAny** returns True if at least one matrix cell satisfies the condition
**_Eigen_ConditCount** returns the number of matrix cells satisfying the condition
These functions also exist column- and row-specific variants.

Neither **Cwise*** nor **Condit*** functions are supported for complex matrices, but they *can* be applied to their real or imaginary parts separately.

Finally, for convenience, the various operator strings can be listed on the fly:
**_Eigen_ShowBinaryOperators**
**_Eigen_ShowConditOperators**
**_Eigen_ShowUnaryOperators**
**_Eigen_ShowScalarOperators**
Note that specific operators may be invalid in some contexts, or applicable to specific types, or ranges, of values only. Invalid operations will usually store an infinity (#INF) or Not-a-Number (#NaN) in the designated cell. Function _Eigen_MatrixSpecs() can test for the presence of these special values (see MatrixSpecs table, spec IDs 28-29). This also implies that **any inadvertent division by zero does not crash your computation, and does not trigger any error!** However, it is the user's responsibility to check that their (intermediate) results are valid.

The following operators are supported for **CwiseBinary** functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

The following operators are supported for **Condit** functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater pr equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

The following operators are supported for **CwiseScalar** functions:

| ID | String | Alias |
|---|---|---|

| | | |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

The following operators are supported for **CwiseUnary** functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer





# Binary Operators

- CwiseBinaryOp
- CwiseBinaryOp_InPlace
- CwiseBinaryOp_Block
- CwiseBinaryOp_Block_InPlace
- CwiseBinaryOp_Col
- CwiseBinaryOp_Col_InPlace
- CwiseBinaryOp_ColCol
- CwiseBinaryOp_ColCol_InPlace
- CwiseBinaryOp_ColRow
- CwiseBinaryOp_ColRow_InPlace
- CwiseBinaryOp_Colwise

- CwiseBinaryOp_Colwise_InPlace
- CwiseBinaryOp_ColwiseCol
- CwiseBinaryOp_ColwiseCol_InPlace
- CwiseBinaryOp_Row
- CwiseBinaryOp_Row_InPlace
- CwiseBinaryOp_RowCol
- CwiseBinaryOp_RowCol_InPlace
- CwiseBinaryOp_RowRow
- CwiseBinaryOp_RowRow_InPlace
- CwiseBinaryOp_Rowwise
- CwiseBinaryOp_Rowwise_InPlace
- CwiseBinaryOp_RowwiseRow
- CwiseBinaryOp_RowwiseRow_InPlace
- Show_BinaryOperators

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |





# _Eigen_CwiseBinaryOp

Apply [operator, value] to each cell of matrix A, using the value in the corresponding cell in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp ( $matA, $operator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

| | |
|---|---|
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_Block*(), *CwiseBinaryOp_Col*(), *CwiseBinaryOp_Colwise*(), *CwiseBinaryOp_Row*(), *CwiseBinaryOp_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )


For $cc = 1 To $binaryOperators[0]

      $matR = _Eigen_CwiseBinaryOp ( $matA, $binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
```

```
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_Block

Apply [operator, value] to each cell of a block in matrix A, using the value in the corresponding cell in a same-sized block in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols, $startRow_input, $startCol_input, $operator, $matB[, $matR = 0
] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $startRow_input | row ID (base-0) of the topmost row of the block in matrix B |
| $startCol_input | column ID (base-0) of the leftmost col of the block in matrix B |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

| | |
|---|---|
| Success: | $matR |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Matrices **A** and **B** do not have to have the same dimensions, but the entire block has to fit inside each matrix. Note that top-left coordinates are to be supplied separately for matrix **B**, and thus may differ from those in matrix **A**.

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp(), CwiseBinaryOp_Col(), CwiseBinaryOp_Colwise(), CwiseBinaryOp_Row(), CwiseBinaryOp_Rowwise()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2
$startRow_input = 1     ; input block's top-left corner
$startCol_input = 1

For $cc = 1 To $binaryOperators[0]

      $matR = _Eigen_CwiseBinaryOp_Block ( $matA, $startRow, $startCol, _
            $blockRows, $blockCols, $startRow_input, $startCol_input, _
            $binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



## CwiseBinaryOp_Block_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_Block_InPlace

Apply [operator, value] to each cell of a block in matrix A, using the value in the corresponding cell in a same-sized block in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Block_InPlace ( $matA, $startRow, $startCol,
$blockRows, $blockCols, $startRow_input, $startCol_input, $operator, $matB
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $startRow_input | row ID (base-0) of the topmost row of the block in matrix B |
| $startCol_input | column ID (base-0) of the leftmost col of the block in matrix B |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Matrices **A** and **B** do not have to have the same dimensions, but the entire block has to fit inside each matrix. Note that top-left coordinates are to be supplied separately for matrix **B**, and thus may differ from those in matrix **A**.

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |

| | | |
|---|---|---|
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace(), CwiseBinaryOp_Col_InPlace(), CwiseBinaryOp_Colwise_InPlace(), CwiseBinaryOp_Row_InPlace(), CwiseBinaryOp_Rowwise_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2
$startRow_input = 1     ; input block's top-left corner
$startCol_input = 1

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_Block_InPlace ( $matA, $startRow, $startCol, _
            $blockRows, $blockCols, $startRow_input, $startCol_input, _
            $binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_Col

Function Reference

# _Eigen_CwiseBinaryOp_Col

Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Col ( $matA, $colindex, $operator, $matB[, $matR =
0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon in both matrices |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp*(), *CwiseBinaryOp_Block*(), *CwiseBinaryOp_Colwise*(), *CwiseBinaryOp_Row*(), *CwiseBinaryOp_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()
```

```
$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$colindex = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_Col ( $matA, $colindex,
$binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_Col_InPlace

Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Col_InPlace ( $matA, $colindex, $operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon in both matrices |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex = 2

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_Col_InPlace ( $matA, $colindex,
$binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & " B" )
Next

_Eigen_CleanUp()
```



CwiseBinaryOp_ColCol

# _Eigen_CwiseBinaryOp_ColCol

Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in another column in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_ColCol ( $matA, $colindex_src, $colindex_dst,
$operator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix B to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(),

*CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$colindex_src = 0
$colindex_dst = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_ColCol ( $matA, $colindex_src,
$colindex_dst, _
           $binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_ColCol_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_ColCol_InPlace

Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in another column in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_ColCol_InPlace ( $matA, $colindex_src, $colindex_dst,
$operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

| | |
|---|---|
| $colindex_src | column ID (base-0) of the column in matrix B to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data
```

```
$colindex_src = 0
$colindex_dst = 2

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_ColCol_InPlace ( $matA, $colindex_src,
$colindex_dst, _
            $binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_ColRow

Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in a column in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_ColRow ( $matA, $colindex_src, $rowindex_dst,
$operator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix B to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$colindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_ColRow ( $matA, $colindex_src,
$rowindex_dst, _
          $binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_ColRow_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_ColRow_InPlace

Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in a column in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_ColRow_InPlace ( $matA, $colindex_src, $rowindex_dst,
$operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix B to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColCol_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_ColRow_InPlace ( $matA, $colindex_src,
$rowindex_dst, _
            $binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_Colwise

Function Reference

# _Eigen_CwiseBinaryOp_Colwise

Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in Colvector B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Colwise ( $matA, $operator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

| | |
|---|---|
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrix **B** should be a Colvector with the same number of rows as matrix **A**.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp(), CwiseBinaryOp_Block(), CwiseBinaryOp_Col(), CwiseBinaryOp_Row(), CwiseBinaryOp_Rowwise()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 1 )      ; B = Colvector!
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 8 )
_MatrixDisplay ( $matB, "original B data" )


For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_Colwise ( $matA, $binaryOperators[$cc],
$matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```

CwiseBinaryOp_Colwise_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_Colwise_InPlace

Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in Colvector B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Colwise_InPlace ( $matA, $operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrix **B** should be a Colvector with the same number of rows as matrix **A**.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 1 )      ; B = Colvector!
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 8 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data


For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_Colwise_InPlace ( $matA, $binaryOperators[$cc],
$matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_ColwiseCol

Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in a single column in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_ColwiseCol ( $matA, $colindex_src, $operator, $matB[,
$matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix B to obtain the values from |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

| | |
|---|---|
| Success: | $matR |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Matrices **A** and **B** have to have the same number of rows.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )


$colindex_src = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_ColwiseCol ( $matA, $colindex_src,
$binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```

# _Eigen_CwiseBinaryOp_ColwiseCol_InPlace

Apply [operator, value] to each cell of matrix A, acting on each column in turn, using the values in the corresponding cells in a single column in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_ColwiseCol_InPlace ( $matA, $colindex_src, $operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix B to obtain the values from |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the same number of rows.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(),

*CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex_src = 2

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_ColwiseCol_InPlace ( $matA, $colindex_src,
$binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_InPlace

Apply [operator, value] to each cell of matrix A, using the value in the corresponding cell in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_InPlace ( $matA, $operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |

| | |
|---|---|
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data


For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_InPlace ( $matA, $binaryOperators[$cc], $matB )
```

```
    _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_Row

Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Row ( $matA, $rowindex, $operator, $matB[, $matR =
0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon in both matrices |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

| | | |
|---|---|---|
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp*(), *CwiseBinaryOp_Block*(), *CwiseBinaryOp_Col*(), *CwiseBinaryOp_Colwise*(), *CwiseBinaryOp_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )


$rowindex = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_Row ( $matA, $rowindex,
$binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_Row_InPlace

Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Row_InPlace ( $matA, $rowindex, $operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon in both matrices |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace(), CwiseBinaryOp_Block_InPlace(), CwiseBinaryOp_Col_InPlace(), CwiseBinaryOp_Colwise_InPlace(), CwiseBinaryOp_Rowwise_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )
```

```
$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex = 2

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_Row_InPlace ( $matA, $rowindex,
$binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_RowCol

Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in one row in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_RowCol ( $matA, $rowindex_src, $colindex_dst,
$operator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix B to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )


$rowindex_src = 0
$colindex_dst = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_RowCol ( $matA, $rowindex_src, $colindex_dst, _
           $binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & " B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_RowCol_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_RowCol_InPlace

Apply [operator, value] to each cell of one column in matrix A, using the value in the corresponding cell in one row in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_RowCol_InPlace ( $matA, $rowindex_src, $colindex_dst,
$operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix B to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_ColCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex_src = 0
$colindex_dst = 2

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_RowCol_InPlace ( $matA, $rowindex_src,
$colindex_dst, _
            $binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_RowRow

Function Reference

# _Eigen_CwiseBinaryOp_RowRow

Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in a row in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_RowRow ( $matA, $rowindex_src, $rowindex_dst,
$operator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix B to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )
```

```
$rowindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_RowRow ( $matA, $rowindex_src,
$rowindex_dst, _
           $binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



## CwiseBinaryOp_RowRow_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_RowRow_InPlace

Apply [operator, value] to each cell of one row in matrix A, using the value in the corresponding cell in a row in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_RowRow_InPlace ( $matA, $rowindex_src, $rowindex_dst,
$operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix B to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_ColCol_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_RowRow_InPlace ( $matA, $rowindex_src,
$rowindex_dst, _
            $binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next
```

```
_Eigen_CleanUp()
```





# _Eigen_CwiseBinaryOp_Rowwise

Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in Rowvector B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Rowwise ( $matA, $operator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrix **B** should be a Rowvector with the same number of columns as matrix **A**.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp*(), *CwiseBinaryOp_Block*(), *CwiseBinaryOp_Col*(), *CwiseBinaryOp_Colwise*(), *CwiseBinaryOp_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 1, 4 )      ; B = Rowvector!
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 8 )
_MatrixDisplay ( $matB, "original B data" )


For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_Rowwise ( $matA, $binaryOperators[$cc],
$matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_Rowwise_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_Rowwise_InPlace

Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in Rowvector B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_Rowwise_InPlace ( $matA, $operator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Matrix **B** should be a Rowvector with the same number of columns as matrix **A**.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 1, 4 )      ; B = Rowvector!
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 8 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data


For $cc = 1 To $binaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_Rowwise_InPlace ( $matA, $binaryOperators[$cc],
$matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```

CwiseBinaryOp_RowwiseRow

Function Reference

# _Eigen_CwiseBinaryOp_RowwiseRow

Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in a single row in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_RowwiseRow ( $matA, $rowindex_src, $operator, $matB[,
$matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix B to obtain the values from |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the same number of columns.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp_InPlace(), CwiseBinaryOp_Block_InPlace(), CwiseBinaryOp_Col_InPlace(),*

*CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_RowwiseRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )


$rowindex_src = 2

For $cc = 1 To $binaryOperators[0]

     $matR = _Eigen_CwiseBinaryOp_RowwiseRow ( $matA, $rowindex_src,
$binaryOperators[$cc], $matB )

     _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```



### CwiseBinaryOp_RowwiseRow_InPlace

Function Reference

# _Eigen_CwiseBinaryOp_RowwiseRow_InPlace

Apply [operator, value] to each cell of matrix A, acting on each row in turn, using the values in the corresponding cells in a single row in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseBinaryOp_RowwiseRow_InPlace ( $matA, $rowindex_src, $operator,
$matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix B to obtain the values from |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | matrix ID of the matrix supplying the value for each cell |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the same number of columns.

The following operators are supported for this specific CwiseBinary function:

| ID | String | Alias |
|---|---|---|
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |

## Related

*CwiseBinaryOp_InPlace*(), *CwiseBinaryOp_Block_InPlace*(), *CwiseBinaryOp_Col_InPlace*(), *CwiseBinaryOp_Colwise_InPlace*(), *CwiseBinaryOp_Row_InPlace*(), *CwiseBinaryOp_Rowwise_InPlace*(), *CwiseBinaryOp_ColRow_InPlace*(), *CwiseBinaryOp_ColwiseCol_InPlace*(), *CwiseBinaryOp_RowRow_InPlace*(), *CwiseBinaryOp_RowCol_InPlace*(), *CwiseBinaryOp_ColCol_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )
```

```
$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex_src = 2

For $cc = 1 To $binaryOperators[0]

      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseBinaryOp_RowwiseRow_InPlace ( $matA, $rowindex_src,
$binaryOperators[$cc], $matB )

      _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$cc] & "
B" )
Next

_Eigen_CleanUp()
```





# _Eigen_Show_BinaryOperators

Display the list of Cwise binary operators

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_BinaryOperators ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

The returned list does not include the various accepted aliases.

## Related

*Show_Conditoperators*(), *Show_ScalarOperators*(), *Show_UnaryOperators*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_BinaryOperators()

_Eigen_CleanUp()
```





# Conditional Operators

- ConditAll
- ConditAll_Col
- ConditAll_Row
- ConditAny
- ConditAny_Col
- ConditAny_Row
- ConditBinaryOp
- ConditBinaryOp_InPlace
- ConditCount
- ConditCount_Col
- ConditCount_Row
- ConditScalarOp
- ConditScalarOp_InPlace
- ConditUnaryOp
- ConditUnaryOp_InPlace
- Show_ConditOperators

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |



## ConditAll

Function Reference

# _Eigen_ConditAll

Evaluate whether *all* cells of matrix A satisfy [condition, scalar]

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditAll ( $matA, $conditOperator, $conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

Success: True/False

Failure: sets the @error flag to non-zero

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll_Col*(), *ConditAll_Row*(), *ConditAny*(), *ConditAny_Col*(), *ConditAny_Row*(), *ConditCount*(), *ConditCount_Col*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
```

```
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 1

For $cc = 1 To $conditOperators[0]
     _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

     MsgBox ( 0, "Do ALL cells satisfy", "condition: " & _
           $conditOperators[$cc] & $conditscalar & "?  " & _
           _Eigen_ConditAll ( $matA, $conditOperators[$cc], $conditScalar ) )
Next

_Eigen_CleanUp()
```





# _Eigen_ConditAll_Col

Evaluate whether *all* cells of a column in matrix A satisfy [condition, scalar]

```
  #include <Eigen4AutoIt.au3>
_Eigen_ConditAll_Col ( $matA, $colindex, $conditOperator, $conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

Success: True/False

Failure: sets the @error flag to non-zero

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |

| | | |
|---|---|---|
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll*(), *ConditAll_Row*(), *ConditAny*(), *ConditAny_Col*(), *ConditAny_Row*(), *ConditCount*(), *ConditCount_Col*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex = 1
$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      MsgBox ( 0, "Do ALL col" & $colindex & " cells satisfy", "condition: " &
_
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditAll_Col ( $matA, $colindex, $conditOperators[$cc],
$conditScalar ) )
Next

_Eigen_CleanUp()
```



### ConditAll_Row

Function Reference

# _Eigen_ConditAll_Row

Evaluate whether *all* cells of a row in matrix A satisfy [condition, scalar]

```
  #include <Eigen4AutoIt.au3>
_Eigen_ConditAll_Row ( $matA, $rowindex, $conditOperator, $conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

Success: True/False

Failure: sets the @error flag to non-zero

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll*(), *ConditAll_Col*(), *ConditAny*(), *ConditAny_Col*(), *ConditAny_Row*(), *ConditCount*(), *ConditCount_Col*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex = 1
$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )   ; reload original data
```

```
    MsgBox ( 0, "Do ALL row" & $rowindex & " cells satisfy", "condition: " &
_
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditAll_Row ( $matA, $rowindex, $conditOperators[$cc],
$conditScalar ) )
Next

_Eigen_CleanUp()
```





# _Eigen_ConditAny

Evaluate whether *any* cell of matrix A satisfies [condition, scalar]

```
  #include <Eigen4AutoIt.au3>
_Eigen_ConditAny ( $matA, $conditOperator, $conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

Success: True/False

Failure: sets the @error flag to non-zero

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll*(), *ConditAll_Col*(), *ConditAll_Row*(), *ConditAny_Col*(), *ConditAny_Row*(), *ConditCount*(), *ConditCount_Col*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 1

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      MsgBox ( 0, "Does ANY cell satisfy", "condition: " & _
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditAny ( $matA, $conditOperators[$cc], $conditScalar ) )
Next

_Eigen_CleanUp()
```



### ConditAny_Col

Function Reference

# _Eigen_ConditAny_Col

Evaluate whether *any* cell of a specified column in matrix A satisfies [condition, scalar]

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditAny_Col ( $matA, $colindex, $conditOperator, $conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon |
| $conditOperat | either an operator string (see Remarks) or the numeric ID thereof |

| | |
|---|---|
| or | |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

Success: True/False

Failure: sets the @error flag to non-zero

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll*(), *ConditAll_Col*(), *ConditAll_Row*(), *ConditAny*(), *ConditAny_Row*(), *ConditCount*(), *ConditCount_Col*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex = 1
$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      MsgBox ( 0, "Do ANY col" & $colindex & " cells satisfy", "condition: " &
_
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditAny_Col ( $matA, $colindex, $conditOperators[$cc],
$conditScalar ) )
Next
```

```
_Eigen_CleanUp()
```





# _Eigen_ConditAny_Row

Evaluate whether *any* cell of a specified row in matrix A satisfies [condition, scalar]

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditAny_Row ( $matA, $rowindex, $conditOperator, $conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

| | |
|---|---|
| Success: | True/False |
| Failure: | sets the @error flag to non-zero |

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll*(), *ConditAll_Col*(), *ConditAll_Row*(), *ConditAny*(), *ConditAny_Col*(), *ConditCount*(), *ConditCount_Col*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex = 1
$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      MsgBox ( 0, "Do ANY row" & $rowindex & " cells satisfy", "condition: " &
_
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditAny_Row ( $matA, $rowindex, $conditOperators[$cc],
$conditScalar ) )
Next

_Eigen_CleanUp()
```



### ConditBinaryOp

Function Reference

# _Eigen_ConditBinaryOp

Apply [operator, value] to each cell of matrix A that satisfies [condition, scalar], using the value in the corresponding cell in matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditBinaryOp ( $matA, $conditoperator, $conditScalar,
$binaryOperator, $matB[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

| | |
|---|---|
| $binaryOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | value to apply with the scalar operator |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*ConditScalarOp*(), *ConditUnaryOp*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()
_Eigen_Show_BinaryOperators()
```

```
$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 8

For $cc = 1 To $conditOperators[0]
     MsgBox ( 0, "Currently Testing A with", "Condition: " & _
           $conditOperators[$cc] & $conditscalar )

     For $bc = 1 To $binaryOperators[0]
           _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

           $matR = _Eigen_ConditBinaryOp ( $matA, $conditOperators[$cc],
$conditscalar, _
                 $binaryOperators[$bc], $matB )

           _MatrixDisplay ( $matR, "after Cwise A " & $binaryOperators[$bc] &
" B" )
     Next
Next

_Eigen_CleanUp()
```





# _Eigen_ConditBinaryOp_InPlace

Apply [operator, value] to each cell of matrix A that satisfies [condition, scalar], using the value in the corresponding cell in matrix B, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditBinaryOp_InPlace ( $matA, $conditoperator, $conditScalar,
$binaryOperator, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

| | |
|---|---|
| $binaryOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $matB | value to apply with the scalar operator |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

The following operators are supported for CwiseBinary functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |

## Related

*ConditScalarOp_InPlace(), ConditUnaryOp_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()
_Eigen_Show_BinaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
```

```
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "original B data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      MsgBox ( 0, "Currently Testing A with", "Condition: " & _
            $conditOperators[$cc] & $conditscalar )

      For $bc = 1 To $binaryOperators[0]
            _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

            _Eigen_ConditBinaryOp_InPlace ( $matA, $conditOperators[$cc],
$conditscalar, _
                  $binaryOperators[$bc], $matB )

            _MatrixDisplay ( $matA, "after Cwise A " & $binaryOperators[$bc] &
" B" )
      Next
Next

_Eigen_CleanUp()
```





# _Eigen_ConditCount

Evaluate *how many* cells of matrix A satisfy [condition, scalar]

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditCount ( $matA, $conditOperator, $conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

| | |
|---|---|
| Success: | zero or positive integer: the number of matrix cells that satisfy the specified condition |
| Failure: | False, and sets the @error flag to non-zero |

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

ConditAll(), *ConditAll_Col*(), *ConditAll_Row*(), *ConditAny*(), *ConditAny_Col*(), *ConditAny_Row*(), *ConditCount_Col*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 1

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      MsgBox ( 0, "HOW MANY cells satisfy", "condition: " & _
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditCount ( $matA, $conditOperators[$cc],
$conditScalar ) )
Next

_Eigen_CleanUp()
```



### ConditCount_Col

Function Reference

# _Eigen_ConditCount_Col

Evaluate *how many* cells of a specified column in matrix A satisfy [condition, scalar]

```
 #include <Eigen4AutoIt.au3>
_Eigen_ConditCount_Col ( $matA, $colindex, $conditOperator,
$conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

| | |
|---|---|
| Success: | zero or positive integer: the number of matrix cells that satisfy the specified condition |
| Failure: | False, and sets the @error flag to non-zero |

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll*(), *ConditAll_Col*(), *ConditAll_Row*(), *ConditAny*(), *ConditAny_Col*(), *ConditAny_Row*(), *ConditCount*(), *ConditCount_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()
```

```
$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex = 1
$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      MsgBox ( 0, "HOW MANY col" & $colindex & " cells satisfy", "condition: "
 & _
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditCount_Col ( $matA, $colindex, $conditOperators[$cc],
$conditScalar ) )

Next

_Eigen_CleanUp()
```





# _Eigen_ConditCount_Row

Evaluate *how many* cells of a specified row in matrix A satisfy [condition, scalar]

```
 #include <Eigen4AutoIt.au3>
_Eigen_ConditCount_Row ( $matA, $rowindex, $conditOperator,
$conditScalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |

## Return Value

| | |
|---|---|
| Success: | zero or positive integer: the number of matrix cells that satisfy the specified condition |
| Failure: | False, and sets the @error flag to non-zero |

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

## Related

*ConditAll*(), *ConditAll_Col*(), *ConditAll_Row*(), *ConditAny*(), *ConditAny_Col*(), *ConditAny_Row*(), *ConditCount*(), *ConditCount_Col*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex = 1
$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      MsgBox ( 0, "HOW MANY row" & $rowindex & " cells satisfy", "condition: "
 & _
            $conditOperators[$cc] & $conditscalar & "?  " & _
            _Eigen_ConditCount_Row ( $matA, $rowindex, $conditOperators[$cc],
$conditScalar ) )
Next

_Eigen_CleanUp()
```



### ConditScalarOp

Function Reference

# _Eigen_ConditScalarOp

Apply [operator, scalar] to each cell of matrix A that satisfies [condition, scalar]

```
 #include <Eigen4AutoIt.au3>
_Eigen_ConditScalarOp ( $matA, $conditOperator, $conditScalar,
$scalarOperator, $scalar[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |
| $scalarOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply with the scalar operator |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |

| | | |
|---|---|---|
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*ConditBinaryOp(), ConditUnaryOp()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()
_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 8
$scalar = 2

For $cc = 1 To $conditOperators[0]
     MsgBox ( 0, "Currently Testing A with", "Condition: " & _
           $conditOperators[$cc] & $conditscalar )

     For $sc = 1 To $scalarOperators[0]
           _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

           $matR = _Eigen_ConditScalarOp ( $matA, $conditOperators[$cc],
$conditscalar, _
                 $scalarOperators[$sc], $matB )

           _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$sc] &
 $scalar )
     Next
Next

_Eigen_CleanUp()
```



### ConditScalarOp_InPlace

Function Reference

# _Eigen_ConditScalarOp_InPlace

Apply [operator, scalar] to each cell of matrix A that satisfies [condition, scalar], storing the

result in-place

```
 #include <Eigen4AutoIt.au3>
_Eigen_ConditScalarOp_InPlace ( $matA, $conditOperator, $conditScalar,
$scalarOperator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |
| $scalarOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply with the scalar operator |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*ConditBinaryOp_InPlace(), ConditUnaryOp_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()
_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 8
$scalar = 2

For $cc = 1 To $conditOperators[0]
      MsgBox ( 0, "Currently Testing A with", "Condition: " & _
            $conditOperators[$cc] & $conditscalar )

      For $sc = 1 To $scalarOperators[0]
            _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

            _Eigen_ConditScalarOp_InPlace ( $matA, $conditOperators[$cc],
$conditscalar, _
                  $scalarOperators[$sc], $matB )

            _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$sc] &
 $scalar )
      Next
Next

_Eigen_CleanUp()
```



### ConditUnaryOp

Function Reference

# _Eigen_ConditUnaryOp

Apply [operator] to each cell of matrix A that satisfies [condition, scalar]

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditUnaryOp ( $matA, $conditoperator, $conditScalar,
$unaryOperator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |
| $unaryOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3 | "acos" | store result of inverse trigonometric function |
| 4 | "asin" | store result of inverse trigonometric function |
| 5 | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7 | "exp" | store number *e* raised to power <value> |
| 8 | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9 | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11 | "sin" | store result of trigonometric function(radians) |
| 12 | "sqrt" | store <value> raised to the power 0.5 |
| 13 | "tan" | store result of trigonometric function(radians) |

## Related

*ConditBinaryOp(), ConditScalarOp()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()
_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      MsgBox ( 0, "Currently Testing A with", "Condition: " & _
            $conditOperators[$cc] & $conditscalar )

      For $sc = 1 To $unaryOperators[0]
            _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

            $matR = _Eigen_ConditUnaryOp ( $matA, $conditOperators[$cc],
$conditscalar, _
                  $unaryOperators[$sc] )

            _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$sc] )
      Next
Next

_Eigen_CleanUp()
```



### ConditUnaryOp_InPlace

Function Reference

# _Eigen_ConditUnaryOp_InPlace

Apply [operator] to each cell of matrix A that satisfies [condition, scalar], storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_ConditUnaryOp_InPlace ( $matA, $conditoperator, $conditScalar,
$unaryOperator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $conditOperator | either an operator string (see Remarks) or the numeric ID thereof |
| $conditScalar | value to evaluate with the conditional operator |
| $unaryOperator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for Conditional functions:

| ID | String | Alias |
|---|---|---|
| 1 | "==" | "=", "equal", "equals", "equal to" |
| 2 | "!=" | "not equal", "not equal to", "<>", "#" |
| 3 | "<" | "smaller", "smaller than" |
| 4 | ">" | "greater", "greater than", "larger", "larger than" |
| 5 | "<=" | "smaller or equal", "equal or smaller", "equal or smaller than" |
| 6 | ">=" | "greater or equal", "equal or greater", "equal or greater than", "larger or equal", "equal or larger", "equal or larger than" |

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3 | "acos" | store result of inverse trigonometric function |
| 4 | "asin" | store result of inverse trigonometric function |
| 5 | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7 | "exp" | store number *e* raised to power <value> |
| 8 | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9 | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11 | "sin" | store result of trigonometric function(radians) |
| 12 | "sqrt" | store <value> raised to the power 0.5 |
| 13 | "tan" | store result of trigonometric function(radians) |

## Related

*ConditBinaryOp_InPlace(), ConditScalarOp_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()
_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$conditscalar = 8

For $cc = 1 To $conditOperators[0]
      MsgBox ( 0, "Currently Testing A with", "Condition: " & _
            $conditOperators[$cc] & $conditscalar )

      For $sc = 1 To $unaryOperators[0]
            _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

            _Eigen_ConditUnaryOp_InPlace ( $matA, $conditOperators[$cc],
$conditscalar, _
                  $unaryOperators[$sc] )

            _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$sc] )
      Next
Next

_Eigen_CleanUp()
```



### Show_ConditOperators

Function Reference

# _Eigen_Show_ConditOperators

Display the list of Cwise conditional operators

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_ConditOperators ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

The returned list does not include the various accepted aliases.

## Related

*Show_Binaryoperators*(), *Show_ScalarOperators*(), *Show_UnaryOperators*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ConditOperators()

_Eigen_CleanUp()
```





# Scalar Operators

- CwiseScalarOp
- CwiseScalarOp_InPlace
- CwiseScalarOp_Block
- CwiseScalarOp_Block_InPlace
- CwiseScalarOp_Col
- CwiseScalarOp_Col_InPlace
- CwiseScalarOp_ColCol
- CwiseScalarOp_ColCol_InPlace
- CwiseScalarOp_ColRow
- CwiseScalarOp_ColRow_InPlace
- CwiseScalarOp_Row
- CwiseScalarOp_Row_InPlace
- CwiseScalarOp_RowCol
- CwiseScalarOp_RowCol_InPlace
- CwiseScalarOp_RowRow
- CwiseScalarOp_RowRow_InPlace
- Show_ScalarOperators

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |





# _Eigen_CwiseScalarOp

Apply [operator, scalar] to each cell of matrix A

```
 #include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp ( $matA, $operator, $scalar[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|

| | | |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_Block*(), *CwiseScalarOp_Col*(), *CwiseScalarOp_Col_Col*(), *CwiseScalarOp_Col_Row*(), *CwiseScalarOp_Row*(), *CwiseScalarOp_Row_Col*(), *CwiseScalarOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$scalar = 2

For $cc = 1 To $scalarOperators[0]

     $matR = _Eigen_CwiseScalarOp ( $matA, $scalarOperators[$cc], $scalar )

     _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```



### CwiseScalarOp_Block

Function Reference

# _Eigen_CwiseScalarOp_Block

Apply [operator, scalar] to each cell of a block in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols, $operator, $scalar[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The entire block has to fit inside the matrix.

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp*(), *CwiseScalarOp_Col*(), *CwiseScalarOp_Col_Col*(), *CwiseScalarOp_Col_Row*(), *CwiseScalarOp_Row*(), *CwiseScalarOp_Row_Col*(), *CwiseScalarOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
```

```
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]

      $matR = _Eigen_CwiseScalarOp_Block ( $matA, $startRow, $startCol, _
            $blockRows, $blockCols, $scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseScalarOp_Block_InPlace

Apply [operator, scalar] to each cell of a block in matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_Block_InPlace ( $matA, $startRow, $startCol,
$blockRows, $blockCols, $operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The entire block has to fit inside the matrix.

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_InPlace*(), *CwiseScalarOp_Col_InPlace*(), *CwiseScalarOp_ColCol_InPlace*(), *CwiseScalarOp_ColRow_InPlace*(), *CwiseScalarOp_Row_InPlace*(), *CwiseScalarOp_RowCol_InPlace*(), *CwiseScalarOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseScalarOp_Block_InPlace ( $matA, $startRow, $startCol, _
            $blockRows, $blockCols, $scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next
```

```
_Eigen_CleanUp()
```





# _Eigen_CwiseScalarOp_Col

Apply [operator, scalar] to each cell of one column in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_Col ( $matA, $colindex, $operator, $scalar[, $matR =
0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp*(), *CwiseScalarOp_Block*(), *CwiseScalarOp_Col_Col*(), *CwiseScalarOp_Col_Row*(), *CwiseScalarOp_Row*(), *CwiseScalarOp_Row_Col*(), *CwiseScalarOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$colindex = 1
$scalar = 2

For $cc = 1 To $scalarOperators[0]

     $matR = _Eigen_CwiseScalarOp_Col ( $matA, $colindex,
$scalarOperators[$cc], $scalar )

     _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```



### CwiseScalarOp_Col_InPlace

Function Reference

# _Eigen_CwiseScalarOp_Col_InPlace

Apply [operator, scalar] to each cell of one column in matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_Col_InPlace ( $matA, $colindex, $operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_InPlace*(), *CwiseScalarOp_Block_InPlace*(), *CwiseScalarOp_ColCol_InPlace*(), *CwiseScalarOp_ColRow_InPlace*(), *CwiseScalarOp_Row_InPlace*(), *CwiseScalarOp_RowCol_InPlace*(), *CwiseScalarOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex = 1
$scalar = 2

For $cc = 1 To $scalarOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseScalarOp_Col_InPlace ( $matA, $colindex,
$scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next
```

```
_Eigen_CleanUp()
```





# _Eigen_CwiseScalarOp_ColCol

Apply [operator, scalar] to each cell of one column in matrix A, storing the result in another column in results matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_ColCol ( $matA, $colindex_src, $colindex_dst,
$operator, $scalar[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp*(), *CwiseScalarOp_Block*(), *CwiseScalarOp_Col*(), *CwiseScalarOp_ColRow*(), *CwiseScalarOp_Row*(), *CwiseScalarOp_RowCol*(), *CwiseScalarOp_RowRow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$colindex_src = 0
$colindex_dst = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]

     $matR = _Eigen_CwiseScalarOp_ColCol ( $matA, $colindex_src,
$colindex_dst, _
           $scalarOperators[$cc], $scalar )

     _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```



### CwiseScalarOp_ColCol_InPlace

Function Reference

# _Eigen_CwiseScalarOp_ColCol_InPlace

Apply [operator, scalar] to each cell of one column in matrix A, storing the result in another column in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_ColCol_InPlace ( $matA, $colindex_src, $colindex_dst,
$operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_InPlace(), CwiseScalarOp_Block_InPlace(), CwiseScalarOp_Col_InPlace(), CwiseScalarOp_ColRow_InPlace(), CwiseScalarOp_Row_InPlace(), CwiseScalarOp_RowCol_InPlace(), CwiseScalarOp_RowRow_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex_src = 0
$colindex_dst = 2

$scalar = 2
```

```
For $cc = 1 To $scalarOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseScalarOp_ColCol_InPlace ( $matA, $colindex_src,
$colindex_dst, _
            $scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseScalarOp_ColRow

Apply [operator, scalar] to each cell of one column in matrix A, storing the result in a row inresults  matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_ColRow ( $matA, $colindex_src, $rowindex_dst,
$operator, $scalar[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp*(), *CwiseScalarOp_Block*(), *CwiseScalarOp_Col*(), *CwiseScalarOp_ColCol*(), *CwiseScalarOp_Row*(), *CwiseScalarOp_RowCol*(), *CwiseScalarOp_RowRow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$colindex_src = 0
$rowindex_dst = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]

     $matR = _Eigen_CwiseScalarOp_ColRow ( $matA, $colindex_src,
$rowindex_dst, _
          $scalarOperators[$cc], $scalar )

     _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```



### CwiseScalarOp_ColRow_InPlace

Function Reference

# _Eigen_CwiseScalarOp_ColRow_InPlace

Apply [operator, scalar] to each cell of one column in matrix A, storing the result in a row in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_ColRow_InPlace ( $matA, $colindex_src, $rowindex_dst,
$operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_InPlace*(), *CwiseScalarOp_Block_InPlace*(), *CwiseScalarOp_Col_InPlace*(), *CwiseScalarOp_ColCol_InPlace*(), *CwiseScalarOp_Row_InPlace*(), *CwiseScalarOp_RowCol_InPlace*(), *CwiseScalarOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex_src = 0
$rowindex_dst = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseScalarOp_ColRow_InPlace ( $matA, $colindex_src,
$rowindex_dst, _
            $scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseScalarOp_InPlace

Apply [operator, scalar] to each cell of matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_InPlace ( $matA, $operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Matrices **A** and **B** have to have the **same** dimensions (rows *and* cols).

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_Block_InPlace*(), *CwiseScalarOp_Col_InPlace*(), *CwiseScalarOp_ColCol_InPlace*(), *CwiseScalarOp_ColRow_InPlace*(), *CwiseScalarOp_Row_InPlace*(), *CwiseScalarOp_RowCol_InPlace*(), *CwiseScalarOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$scalar = 2

For $cc = 1 To $scalarOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseScalarOp_InPlace ( $matA, $scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```



### CwiseScalarOp_Row

Function Reference

# _Eigen_CwiseScalarOp_Row

Apply [operator, scalar] to each cell of one row in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_Row ( $matA, $rowindex, $operator, $scalar[, $matR =
0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp*(), *CwiseScalarOp_Block*(), *CwiseScalarOp_Col*(), *CwiseScalarOp_Col_Col*(), *CwiseScalarOp_Col_Row*(), *CwiseScalarOp_Row_Col*(), *CwiseScalarOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$rowindex = 1
$scalar = 2

For $cc = 1 To $scalarOperators[0]

     $matR = _Eigen_CwiseScalarOp_Col ( $matA, $rowindex,
$scalarOperators[$cc], $scalar )

     _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp(
```



## CwiseScalarOp_Row_InPlace

Function Reference

# _Eigen_CwiseScalarOp_Row_InPlace

Apply [operator, scalar] to each cell of one row in matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_Row_InPlace ( $matA, $rowindex, $operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_InPlace*(), *CwiseScalarOp_Block_InPlace*(), *CwiseScalarOp_Col_InPlace*(), *CwiseScalarOp_ColCol_InPlace*(), *CwiseScalarOp_ColRow_InPlace*(), *CwiseScalarOp_RowCol_InPlace*(), *CwiseScalarOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex = 1
$scalar = 2

For $cc = 1 To $scalarOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseScalarOp_Col_InPlace ( $matA, $rowindex,
$scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp(
```



### CwiseScalarOp_RowCol

Function Reference

# _Eigen_CwiseScalarOp_RowCol

Apply [operator, scalar] to each cell of one row in matrix A, storing the result in a column in results matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_RowCol ( $matA, $rowindex_src, $colindex_dst,
$operator, $scalar[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp*(), *CwiseScalarOp_Block*(), *CwiseScalarOp_Col*(), *CwiseScalarOp_ColCol*(),

*CwiseScalarOp_ColRow*(), *CwiseScalarOp_Row*(), *CwiseScalarOp_RowRow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$rowindex_src = 0
$colindex_dst = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]

     $matR = _Eigen_CwiseScalarOp_RowCol ( $matA, $rowindex_src,
$colindex_dst, _
          $scalarOperators[$cc], $scalar )

     _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```



### CwiseScalarOp_RowCol_InPlace

Function Reference

# _Eigen_CwiseScalarOp_RowCol_InPlace

Apply [operator, scalar] to each cell of one row in matrix A, storing the result in a column in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_RowCol_InPlace ( $matA, $rowindex_src, $colindex_dst,
$operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |

| | |
|---|---|
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_InPlace(), CwiseScalarOp_Block_InPlace(), CwiseScalarOp_Col_InPlace(), CwiseScalarOp_ColCol_InPlace(), CwiseScalarOp_ColRow_InPlace(), CwiseScalarOp_Row_InPlace(), CwiseScalarOp_RowRow_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex_src = 0
$colindex_dst = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]
     _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data
```

```
      _Eigen_CwiseScalarOp_RowCol_InPlace ( $matA, $rowindex_src,
$colindex_dst, _
            $scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseScalarOp_RowRow

Apply [operator, scalar] to each cell of one row in matrix A, storing the result in another row in results matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_RowRow ( $matA, $rowindex_src, $rowindex_dst,
$operator, $scalar[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |

| | | |
|---|---|---|
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp*(), *CwiseScalarOp_Block*(), *CwiseScalarOp_Col*(), *CwiseScalarOp_ColCol*(), *CwiseScalarOp_ColRow*(), *CwiseScalarOp_Row*(), *CwiseScalarOp_RowCol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor( $matA, 1, 16 )
_MatrixDisplay ( $matA, "original A data" )

$rowindex_src = 0
$rowindex_dst = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]

     $matR = _Eigen_CwiseScalarOp_RowRow ( $matA, $rowindex_src,
$rowindex_dst, _
          $scalarOperators[$cc], $scalar )

     _MatrixDisplay ( $matR, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```



### CwiseScalarOp_RowRow_InPlace

Function Reference

# _Eigen_CwiseScalarOp_RowRow_InPlace

Apply [operator, scalar] to each cell of one row in matrix A, storing the result in another row in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseScalarOp_RowRow_InPlace ( $matA, $rowindex_src, $rowindex_dst,
$operator, $scalar )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $scalar | value to apply |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseScalar functions:

| ID | String | Alias |
|---|---|---|
| 0 | "replace" | "replace with" |
| 1 | "+" | "plus", "add", "addition", "+=" |
| 2 | "-" | "minus", "sub", "subtraction", "-=" |
| 3 | "/" | "divide", "div", "quotient", "division", "/=" |
| 4 | "*" | "multiply", "mul", "mult", "product", "multiplication", "*=" |
| 5 | "max" | "maximum" |
| 6 | "min" | "minimum" |
| 7 | "pow" | "power", "^", "^=" |

## Related

*CwiseScalarOp_InPlace(), CwiseScalarOp_Block_InPlace(), CwiseScalarOp_Col_InPlace(), CwiseScalarOp_ColCol_InPlace(), CwiseScalarOp_ColRow_InPlace(), CwiseScalarOp_Row_InPlace(), CwiseScalarOp_RowCol_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor( $matA, 1, 16 )
```

```
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex_src = 0
$rowindex_dst = 2

$scalar = 2

For $cc = 1 To $scalarOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseScalarOp_RowRow_InPlace ( $matA, $rowindex_src,
$rowindex_dst, _
            $scalarOperators[$cc], $scalar )

      _MatrixDisplay ( $matA, "after Cwise A " & $scalarOperators[$cc] &
$scalar )
Next

_Eigen_CleanUp()
```





# _Eigen_Show_ScalarOperators

Display the list of Cwise scalar operators

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_ScalarOperators ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

The returned list does not include the various accepted aliases.

## Related

*Show_Binaryoperators*(), *Show_Conditoperators*(), *Show_UnaryOperators*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_ScalarOperators()

_Eigen_CleanUp()
```





# Unary Operators

- CwiseUnaryOp
- CwiseUnaryOp_InPlace
- CwiseUnaryOp_Block
- CwiseUnaryOp_Block_InPlace
- CwiseUnaryOp_Col
- CwiseUnaryOp_Col_InPlace
- CwiseUnaryOp_ColCol
- CwiseUnaryOp_ColCol_InPlace
- CwiseUnaryOp_ColRow
- CwiseUnaryOp_ColRow_InPlace
- CwiseUnaryOp_Row
- CwiseUnaryOp_Row_InPlace
- CwiseUnaryOp_RowCol
- CwiseUnaryOp_RowCol_InPlace
- CwiseUnaryOp_RowRow
- CwiseUnaryOp_RowRow_InPlace
- Show_UnaryOperators

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |

| | | |
|---|---|---|
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer





# _Eigen_CwiseUnaryOp

Apply [operator] to each cell of matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp ( $matA, $operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |

| | | |
|---|---|---|
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_Block*(), *CwiseUnaryOp_Col*(), *CwiseUnaryOp_Col_Col*(), *CwiseUnaryOp_Col_Row*(), *CwiseUnaryOp_Row*(), *CwiseUnaryOp_Row_Col*(), *CwiseUnaryOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

For $cc = 1 To $unaryOperators[0]

     $matR = _Eigen_CwiseUnaryOp ( $matA, $unaryOperators[$cc] )

     _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```



### CwiseUnaryOp_Block

Function Reference

# _Eigen_CwiseUnaryOp_Block

Apply [operator] to each cell of a block in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols, $operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The entire block has to fit inside the matrix.

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp*(), *CwiseUnaryOp_Col*(), *CwiseUnaryOp_Col_Col*(), *CwiseUnaryOp_Col_Row*(), *CwiseUnaryOp_Row*(), *CwiseUnaryOp_Row_Col*(), *CwiseUnaryOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2

For $cc = 1 To $unaryOperators[0]

      $matR = _Eigen_CwiseUnaryOp_Block ( $matA, $startRow, $startCol, _
            $blockRows, $blockCols, $unaryOperators[$cc] )

      _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_Block_InPlace

Apply [operator] to each cell of a block in matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_Block_InPlace ( $matA, $startRow, $startCol,
$blockRows, $blockCols, $operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost col of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The entire block has to fit inside the matrix.

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_InPlace*(), *CwiseUnaryOp_Col_InPlace*(), *CwiseUnaryOp_ColCol_InPlace*(), *CwiseUnaryOp_ColRow_InPlace*(), *CwiseUnaryOp_Row_InPlace*(), *CwiseUnaryOp_RowCol_InPlace*(), *CwiseUnaryOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2
```

```
For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_Block_InPlace ( $matA, $startRow, $startCol, _
            $blockRows, $blockCols, $unaryOperators[$cc] )

      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```



## CwiseUnaryOp_Col

Function Reference

# _Eigen_CwiseUnaryOp_Col

Apply [operator] to each cell of one column in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_Col ( $matA, $colindex, $operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |

| | | |
|---|---|---|
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp*(), *CwiseUnaryOp_Block*(), *CwiseUnaryOp_Col_Col*(), *CwiseUnaryOp_Col_Row*(), *CwiseUnaryOp_Row*(), *CwiseUnaryOp_Row_Col*(), *CwiseUnaryOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$colindex = 1

For $cc = 1 To $unaryOperators[0]

     $matR = _Eigen_CwiseUnaryOp_Col ( $matA, $colindex,
$unaryOperators[$cc] )

     _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```



### CwiseUnaryOp_Col_InPlace

Function Reference

# _Eigen_CwiseUnaryOp_Col_InPlace

Apply [operator] to each cell of one column in matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
```

```
_Eigen_CwiseUnaryOp_Col_InPlace ( $matA, $colindex, $operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex | column ID (base-0) of the column to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_InPlace*(), *CwiseUnaryOp_Block_InPlace*(), *CwiseUnaryOp_ColCol_InPlace*(), *CwiseUnaryOp_ColRow_InPlace*(), *CwiseUnaryOp_Row_InPlace*(), *CwiseUnaryOp_RowCol_InPlace*(), *CwiseUnaryOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex = 1

For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_Col_InPlace ( $matA, $colindex,
$unaryOperators[$cc] )

      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_ColCol

Apply [operator] to each cell of a column in matrix A, storing the result in another column in results matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_ColCol ( $matA, $colindex_src, $colindex_dst,
$operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp*(), *CwiseUnaryOp_Block*(), *CwiseUnaryOp_Col*(), *CwiseUnaryOp_Col_Row*(), *CwiseUnaryOp_Row*(), *CwiseUnaryOp_Row_Col*(), *CwiseUnaryOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$colindex_src = 0
$colindex_dst = 2

For $cc = 1 To $unaryOperators[0]

     $matR = _Eigen_CwiseUnaryOp_ColCol ( $matA, $colindex_src,
$colindex_dst, $unaryOperators[$cc] )

     _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```

CwiseUnaryOp_ColCol_InPlace

Function Reference

# _Eigen_CwiseUnaryOp_ColCol_InPlace

Apply [operator] to each cell of a column in matrix A, storing the result in another column in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_ColCol_InPlace ( $matA, $colindex_src, $colindex_dst,
$operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |

| | | |
|---|---|---|
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_InPlace*(), *CwiseUnaryOp_Block_InPlace*(), *CwiseUnaryOp_Col_InPlace*(), *CwiseUnaryOp_ColRow_InPlace*(), *CwiseUnaryOp_Row_InPlace*(), *CwiseUnaryOp_RowCol_InPlace*(), *CwiseUnaryOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex_src = 0
$colindex_dst = 2

For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_ColCol_InPlace ( $matA, $colindex_src,
$colindex_dst, $unaryOperators[$cc] )

      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_ColRow

Apply [operator] to each cell of one column in matrix A, storing the result in a row in results matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_Col_Row ( $matA, $colindex_src, $rowindex_dst,
$operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp*(), *CwiseUnaryOp_Block*(), *CwiseUnaryOp_Col*(), *CwiseUnaryOp_ColCol*(), *CwiseUnaryOp_Row*(), *CwiseUnaryOp_RowCol*(), *CwiseUnaryOp_RowRow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$colindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $unaryOperators[0]

     $matR = _Eigen_CwiseUnaryOp_ColRow ( $matA, $colindex_src,
$rowindex_dst, $unaryOperators[$cc] )

     _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_ColRow_InPlace

Apply [operator] to each cell of one column in matrix A, storing the result in a row in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_ColRow_InPlace ( $matA, $colindex_src, $rowindex_dst,
$operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $colindex_src | column ID (base-0) of the column in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_InPlace*(), *CwiseUnaryOp_Block_InPlace*(), *CwiseUnaryOp_Col_InPlace*(), *CwiseUnaryOp_ColCol_InPlace*(), *CwiseUnaryOp_Row_InPlace*(), *CwiseUnaryOp_RowCol_InPlace*(), *CwiseUnaryOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$colindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_ColRow_InPlace ( $matA, $colindex_src,
$rowindex_dst, $unaryOperators[$cc] )

      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next
```

```
_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_InPlace

Apply [operator] to each cell of matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_InPlace ( $matA, $operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_Block_InPlace(), CwiseUnaryOp_Col_InPlace(), CwiseUnaryOp_ColCol_InPlace(), CwiseUnaryOp_ColRow_InPlace(), CwiseUnaryOp_Row_InPlace(), CwiseUnaryOp_RowCol_InPlace(), CwiseUnaryOp_RowRow_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_InPlace ( $matA, $unaryOperators[$cc] )

      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_Row

Apply [operator] to each cell of one row in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_Row ( $matA, $rowindex, $operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

| | |
|---|---|
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp*(), *CwiseUnaryOp_Block*(), *CwiseUnaryOp_Col*(), *CwiseUnaryOp_Col_Col*(), *CwiseUnaryOp_Col_Row*(), *CwiseUnaryOp_Row_Col*(), *CwiseUnaryOp_Row_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$rowindex = 1

For $cc = 1 To $unaryOperators[0]

     $matR = _Eigen_CwiseUnaryOp_Row ( $matA, $rowindex,
```

```
$unaryOperators[$cc] )

      _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_Row_InPlace

Apply [operator] to each cell of one row in matrix A, storing the result in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_Row_InPlace ( $matA, $rowindex, $operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex | row ID (base-0) of the row to act upon |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |

| | | |
|---|---|---|
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_InPlace*(), *CwiseUnaryOp_Block_InPlace*(), *CwiseUnaryOp_Col_InPlace*(), *CwiseUnaryOp_ColCol_InPlace*(), *CwiseUnaryOp_ColRow_InPlace*(), *CwiseUnaryOp_RowCol_InPlace*(), *CwiseUnaryOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex = 1

For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_Row_InPlace ( $matA, $rowindex,
$unaryOperators[$cc] )

      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```



### CwiseUnaryOp_RowCol

Function Reference

# _Eigen_CwiseUnaryOp_RowCol

Apply [operator] to each cell of one row in matrix A, storing the result in a column in results matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_RowCol ( $matA, $rowindex_src, $colindex_dst,
```

```
$operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp*(), *CwiseUnaryOp_Block*(), *CwiseUnaryOp_Col*(), *CwiseUnaryOp_ColCol*(), *CwiseUnaryOp_ColRow*(), *CwiseUnaryOp_Row*(), *CwiseUnaryOp_RowRow*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$rowindex_src = 0
$colindex_dst = 2

For $cc = 1 To $unaryOperators[0]

     $matR = _Eigen_CwiseUnaryOp_RowCol ( $matA, $rowindex_src,
$colindex_dst, $unaryOperators[$cc] )

     _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```





# _Eigen_CwiseUnaryOp_RowCol_InPlace

Apply [operator] to each cell of one row in matrix A, storing the result in a column in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_RowCol_InPlace ( $matA, $rowindex_src, $colindex_dst,
$operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $colindex_dst | column ID (base-0) of the column in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix has to be **square**.

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_InPlace*(), *CwiseUnaryOp_Block_InPlace*(), *CwiseUnaryOp_Col_InPlace*(), *CwiseUnaryOp_ColCol_InPlace*(), *CwiseUnaryOp_ColRow_InPlace*(), *CwiseUnaryOp_Row_InPlace*(), *CwiseUnaryOp_RowRow_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex_src = 0
$colindex_dst = 2

For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_RowCol_InPlace ( $matA, $rowindex_src,
$colindex_dst, $unaryOperators[$cc] )
```

```
      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```



CwiseUnaryOp_RowRow

Function Reference

# _Eigen_CwiseUnaryOp_RowRow

Apply [operator] to each cell of one row in matrix A, storing the result in another row in results matrix R

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_RowRow ( $matA, $rowindex_src, $rowindex_dst,
$operator[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |

| | | |
|---|---|---|
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related <<<<

*CwiseUnaryOp*(), *CwiseUnaryOp_Block*(), *CwiseUnaryOp_Col*(), *CwiseUnaryOp_ColCol*(), *CwiseUnaryOp_ColRow*(), *CwiseUnaryOp_Row*(), *CwiseUnaryOp_RowCol*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$rowindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $unaryOperators[0]

     $matR = _Eigen_CwiseUnaryOp_RowRow ( $matA, $rowindex_src,
$rowindex_dst, $unaryOperators[$cc] )

     _MatrixDisplay ( $matR, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```



### CwiseUnaryOp_RowRow_InPlace

Function Reference

# _Eigen_CwiseUnaryOp_RowRow_InPlace

Apply [operator] to each cell of one row in matrix A, storing the result in another row in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_CwiseUnaryOp_RowRow_InPlace ( $matA, $rowindex_src, $rowindex_dst,
$operator )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the matrix to act upon |
| $rowindex_src | row ID (base-0) of the row in matrix A to obtain the values from |
| $rowindex_dst | row ID (base-0) of the row in matrix A to store the result |
| $operator | either an operator string (see Remarks) or the numeric ID thereof |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The following operators are supported for CwiseUnary functions (no aliases are accepted):

| ID | String | Explanation |
|---|---|---|
| 1 | "abs" | store absolute <value> |
| 2 | "square" | store <value> raised to the power 2 |
| 3* | "acos" | store result of inverse trigonometric function |
| 4* | "asin" | store result of inverse trigonometric function |
| 5* | "cos" | store result of trigonometric function(radians) |
| 6 | "cube" | store the value raised to the power 3 |
| 7* | "exp" | store number *e* raised to power <value> |
| 8* | "inverse" | store 1/<value> cellwise; do not confuse with matrix inverse! |
| 9* | "log" | store natural log (base e) of <value> |
| 10 | "-" | store -<value> (flip sign) |
| 11* | "sin" | store result of trigonometric function(radians) |
| 12* | "sqrt" | store <value> raised to the power 0.5 |
| 13* | "tan" | store result of trigonometric function(radians) |

* = unavailable for matrices of type integer

## Related

*CwiseUnaryOp_InPlace*(), *CwiseUnaryOp_Block_InPlace*(), *CwiseUnaryOp_Col_InPlace*(), *CwiseUnaryOp_ColCol_InPlace*(), *CwiseUnaryOp_ColRow_InPlace*(), *CwiseUnaryOp_Row_InPlace*(), *CwiseUnaryOp_RowCol_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 0, 15 )
_MatrixDisplay ( $matA, "original A data" )

$matC = _Eigen_CloneMatrix ( $matA )      ; buffer to reload original data

$rowindex_src = 0
$rowindex_dst = 2

For $cc = 1 To $unaryOperators[0]
      _Eigen_Copy_A_toB ( $matC, $matA )  ; reload original data

      _Eigen_CwiseUnaryOp_RowRow_InPlace ( $matA, $rowindex_src,
$rowindex_dst, $unaryOperators[$cc] )

      _MatrixDisplay ( $matA, "after Cwise A " & $unaryOperators[$cc] )
Next

_Eigen_CleanUp()
```



### Show_UnaryOperators

Function Reference

# _Eigen_Show_UnaryOperators

Display the list of Cwise unary operators

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_UnaryOperators ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Failure would indicate that **Eigen4AutoIt**'s work environment was not properly initialised.

The returned list does not include the various accepted aliases.

## Related

*Show_Binaryoperators*(), *Show_Conditoperators*(), *Show_ScalarOperators*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_UnaryOperators()

_Eigen_CleanUp()
```



### Reduction

# Matrix Reduction

Reduction "reduces" the content of a matrix to one or several characteristic values. The most important function is **_Eigen_MatrixSpecs (** $matA **)**; it returns an array of over twenty reals, integers, and booleans that describe the target in various ways, for example, the sum, product, and mean of all cells, the number of non-zero cells, the location and value of the maximum and minimum, and whether the matrix is upper-triangular, or is filled with ones, or is an Identity or Null matrix.

As usual, we can also apply this function to any matrix part:
**_Eigen_MatrixSpecs_Diag/Col/Row/Block()**

**Some specs are available only for square matrices, or for full matrices and blocks, or for real, or complex matrices only.** For example, computing the matrix *determinant* requires a square, non-integer matrix; trace and most booleans (except IsOnes and IsZero) act only on a full matrix; IsUnitary is relevant only in complex contexts, and norm (computed by taking the square root of the squared norm, always returning a real) is unavailable for matrices of integer type.

These functions all return an array. If we're only interested in obtaining one spec, we can use:
$value = **_Eigen_MatrixSpecs_Single (** $matA, $specID **)**
with $specID corresponding to the MatrixSpecs array index, and the obtained spec being returned directly. Again, we can alternatively select any matrix subset to work on:

**_Eigen_MatrixSpecs_Diag/Col/Row/Block/Colwise/Rowwise_Single**, and unlike the full-array variety, here we can also act *Colwise* or *Rowwise*, and store the results in a vector (supported specs for Colwise/Rowwise are: 1-3, 5-6, 9-10 only).

**MatrixSpecs*_Single** functions are not supported for complex matrices, but they *can* be applied to their real or imaginary parts separately.

**_Eigen_IsCwiseEqual_AB (** $matA, $matB, $precision **)** offers a different kind of reduction, returning a boolean (True/False) depending on whether all cell-by-cell comparisons between the two matrices yield a difference smaller than $precision (default precision (parsed value: -1) is **1.0e-6** for floats, **1.0e-12** for doubles).
Likewise, **_Eigen_IsCwiseEqual_ToConstant (** $matA, $value, $precision **)** compares each cell's value to a supplied scalar, at the default or specified level of precision.
This additional precision-based control is the main difference with the similar **Cwise**-Conditional function **_Eigen_ConditAll()**, which always performs an exact comparison.

Overview of all **Matrix Specs**:

| ID | MatrixSpec | Remarks |
|---|---|---|
| 0 | Trace | returns real (float or double); matrix only |
| 1 | Sum | returns real |
| 2 | Product | returns real |
| 3 | Mean | returns real |
| 4 | Determinant | returns real; square non-integer matrix only |
| 5 | Norm | returns real; unavailable for integer matrix |
| 6 | Squared norm | returns real |
| 7 | LpNorm<1> | returns real |
| 8 | LpNorm<Infinity> | returns real |
| 9 | Minimum value | returns real |
| 10 | Maximum value | returns real |
| 11 | Minimum value's row | returns integer; row coordinate of minimum value (ID 9) |
| 12 | Minimum's value's column | returns integer; col coordinate of minimum value (ID 9) |
| 13 | Maximum value's row | returns integer; row coordinate of maximum value (ID 10) |
| 14 | Maximum value's column | returns integer; col coordinate of maximum value (ID 10) |
| 15 | Size | returns integer; number of cells |
| 16 | nonZeroes | returns integer; number of cells that ever held a value |
| 17 - 22 | *<reserved>* | *to be defined* |
| 23 | IsDiagonal | returns boolean; matrix only |
| 24 | IsIdentity | returns boolean; matrix only |
| 25 | IsLowerTriangular | returns boolean; matrix only |
| 26 | IsUpperTriangular | returns boolean; matrix only |
| 27 | IsUnitary | returns boolean; complex matrix only |
| 28 | AllFinite | returns boolean |
| 29 | hasNaN | returns boolean |
| 30 | IsOnes | returns boolean |
| 31 | IsZero | returns boolean |



## IsCwiseEqual_AB

Function Reference

# _Eigen_IsCwiseEqual_AB

Evaluate whether input matrices A and B are equal, to within some small margin

```
#include <Eigen4AutoIt.au3>
_Eigen_IsCwiseEqual_AB ( $matA, $matB[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: True / False

Failure: False, and sets the @error flag to non-zero.

## Remarks

The two input matrices have to have the same dimensions.

The comparison returns True if no cell-wise absolute difference exceeds $epsilon.

## Related

*IsCwiseEqual_ToConstant()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Ones ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CloneMatrix ( $matA )      ; identical
_MatrixDisplay ( $matB, "matrix B = A" )

$matC = _Eigen_CloneMatrix ( $matA )
_Eigen_WriteMatrixValue ( $matC, 2, 2, 999 )    ; one cell is different
_MatrixDisplay ( $matA, "matrix C <> A" )

MsgBox ( 0, "IsCwiseEqual", "Is A = B ?  " & _
```

```
    _Eigen_IsCwiseEqual_AB ( $matA, $matB ))

MsgBox ( 0, "IsCwiseEqual", "Is A = C ?  " & _
    _Eigen_IsCwiseEqual_AB ( $matA, $matC ))

_Eigen_CleanUp()
```





# _Eigen_IsCwiseEqual_ToConstant

Evaluate whether all cells of input matrix A are equal to a constant, to within some small margin

```
#include <Eigen4AutoIt.au3>
_Eigen_IsCwiseEqual_ToConstant ( $matA, $value[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $value | constant to compare the cell contents with |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: True / False

Failure: False, and sets the @error flag to non-zero.

## Remarks

The comparison returns True if no absolute difference between cell content and the supplied constant exceeds $epsilon.

## Related

*IsCwiseEqual_AB()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
$matA = _Eigen_CreateMatrix_Ones ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$value = 1
MsgBox ( 0, "IsCwiseEqual", "Is A = " & $value & " ?  " & _
      _Eigen_IsCwiseEqual_ToConstant ( $matA, $value ) )

$value = 2
MsgBox ( 0, "IsCwiseEqual", "Is A = " & $value & " ?  " & _
      _Eigen_IsCwiseEqual_ToConstant ( $matA, $value ) )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs

Evaluate and return a range of Matrix Specs for a matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs ( $matA[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: array (32 x 2) containing all matrix specs (labels in column 0, values in column 1)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs_Block*(), *MatrixSpecs_Col*(), *MatrixSpecs_Diag*(), *MatrixSpecs_Row*(), *MatrixSpecs_Single*(), Show_*MatrixSpecs*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_ArrayDisplay ( _MatrixSpecs ( $matA ) )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Block

Evaluate and return a range of Matrix Specs for a block

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: array (32 x 2) containing all matrix specs (labels in column 0, values in column 1)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Col*(), *MatrixSpecs_Diag*(), *MatrixSpecs_Row*(), *MatrixSpecs_Single*(), Show_*MatrixSpecs*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )

_MatrixDisplay ( $matA, "matrix A" )

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 3    ; size of target block
$blockCols = 3

_ArrayDisplay ( _MatrixSpecs_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols ) )

_Eigen_CleanUp()
```



### MatrixSpecs_Block_Single

Function Reference

# _Eigen_MatrixSpecs_Block_Single

Evaluate and return a single Matrix Spec for a defined block

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Block_Single ( $matA, $startRow, $startCol, $blockRows,
$blockCols, $specID[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $startRow | row ID (base-0) of the topmost row of the block in target matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in target matrix A |
| $blockRows | block row dimension |
| $blockCols | block column dimension |
| $specID | ID (0-31) of the desired Matrix Spec to be returned |

| | |
|---|---|
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: the requested MatrixSpec value (real, integer, or boolean)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Single*(), *MatrixSpecs_Col_Single*(), *MatrixSpecs_Diag_Single*(), *MatrixSpecs_Row_Single*(), *MatrixSpecs_Colwise_Single*(), *MatrixSpecs_Rowwise_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )

_MatrixDisplay ( $matA, "matrix A" )

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 3    ; size of target block
$blockCols = 3

$specID = 3       ; ID 3 = mean (average)

 MsgBox ( 0, "Mean value of cells in", "defined block = " & _
      _MatrixSpecs_Block_Single ( $matA, $startRow, $startCol, _
            $blockRows, $blockCols, $specID ) )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Col

Evaluate and return a range of Matrix Specs for one column

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Col ( $matA, $colindex[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $colindex | column ID (base-0) of the column to act upon |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: array (32 x 2) containing all matrix specs (labels in column 0, values in column 1)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

Several matrixSpecs do not apply to vectors (row, col, diagonal).

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Block*(), *MatrixSpecs_Diag*(), *MatrixSpecs_Row*(), *MatrixSpecs_Single*(), Show*_MatrixSpecs*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$colindex = 2
_ArrayDisplay ( _MatrixSpecs_Col ( $matA, $colindex ) )

_Eigen_CleanUp()
```



### MatrixSpecs_Col_Single

Function Reference

# _Eigen_MatrixSpecs_Col_Single

Evaluate and return a single Matrix Spec for one column

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Col_Single ( $matA, $colindex, $specID[, $epsilon = -1]
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $colindex | column ID (base-0) of the column to act upon |
| $specID | ID (0-31) of the desired Matrix Spec to be returned |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: the requested MatrixSpec value (real, integer, or boolean)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

Several MatrixSpecs do not apply to vectors (row, col, diagonal).

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Single*(), *MatrixSpecs_Block_Single*(), *MatrixSpecs_Diag_Single*(), *MatrixSpecs_Row_Single*(), *MatrixSpecs_Colwise_Single*(), *MatrixSpecs_Rowwise_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$colindex = 2
$specID = 3       ; ID 3 = mean (average)
```

```
MsgBox ( 0, "Mean value of cells in", "column " & $colindex & " = " & _
     _MatrixSpecs_Col_Single ( $matA, $colindex, $specID ) )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Colwise_Single

Evaluate and return a specified Matrix Spec for each column; store the result in a Rowvector

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Colwise_Single ( $matA, $specID[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $specID | ID (0-31) of the desired Matrix Spec to be returned |
| $matR | **[optional]** matrix ID of the results matrix (vector) |

## Return Value

| | |
|---|---|
| Success: | array (32 x 2) containing all matrix specs (labels in column 0, values in column 1) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

This function does not accept any matrix inputs of, or related to, complex types.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, a Rowvector with the same number of columns as input matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

Only a small subset of MatrixSpecs is supported here (none of which involves a precision-dependent evaluation), viz.

| ID | MatrixSpec | Remarks |
|---|---|---|
| 1 | Sum | returns real |
| 2 | Product | returns real |
| 3 | Mean | returns real |
| 5 | Norm | returns real |
| 6 | Squared norm | returns real |
| 9 | Minimum value | returns real |

| | | |
|---|---|---|
| 10 | Maximum value | returns real |

## Related

*MatrixSpecs*(), *MatrixSpecs_Single*(), *MatrixSpecs_Block_Single*(), *MatrixSpecs_Col_Single*(), *MatrixSpecs_Diag_Single*(), *MatrixSpecs_Row_Single*(), *MatrixSpecs_Rowwise_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$specID = 3       ; ID 3 = mean (average)

$matM = _MatrixSpecs_Colwise_Single ( $matA, $specID )
_MatrixDisplay ( $matM, "Colwise means" )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Diag

Evaluate and return a range of Matrix Specs for the matrix diagonal

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Diag ( $matA[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

| | |
|---|---|
| Success: | array (32 x 2) containing all matrix specs (labels in column 0, values in column 1) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

Several matrixSpecs do not apply to vectors (row, col, diagonal).

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Block*(), *MatrixSpecs_Col*(), *MatrixSpecs_Row*(), *MatrixSpecs_Single*(), Show_*MatrixSpecs*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_ArrayDisplay ( _MatrixSpecs_Diag ( $matA ) )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Diag_Single

Evaluate and return a single Matrix Spec for the matrix diagonal

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Diag_Single ( $matA, $specID[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $specID | ID (0-31) of the desired Matrix Spec to be returned |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

| | |
|---|---|
| Success: | the requested MatrixSpec value (real, integer, or boolean) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

Several MatrixSpecs do not apply to vectors (row, col, diagonal).

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Single*(), *MatrixSpecs_Block_Single*(), *MatrixSpecs_Col_Single*(), *MatrixSpecs_Row_Single*(), *MatrixSpecs_Colwise_Single*(), *MatrixSpecs_Rowwise_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2
$specID = 3       ; ID 3 = mean (average)

 MsgBox ( 0, "Mean value of cells in", "diagonal = " & _
     _MatrixSpecs_Diag_Single ( $matA, $specID ) )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Row

Evaluate and return a range of Matrix Specs for one selected row

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Row ( $matA, $rowindex[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |

| | |
|---|---|
| $rowindex | row ID (base-0) of the row to act upon |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

| | |
|---|---|
| Success: | array (32 x 2) containing all matrix specs (labels in column 0, values in column 1) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

Several MatrixSpecs do not apply to vectors (row, col, diagonal).

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Block*(), *MatrixSpecs_Col*(), *MatrixSpecs_Diag*(), *MatrixSpecs_Single*(), Show_*MatrixSpecs*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2
_ArrayDisplay ( _MatrixSpecs_Row ( $matA, $rowindex ) )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Row_Single

Evaluate and return a single Matrix Spec for one selected row

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Row_Single ( $matA, $rowindex, $specID[, $epsilon = -1]
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $rowindex | row ID (base-0) of the row to act upon |
| $specID | ID (0-31) of the desired Matrix Spec to be returned |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: the requested MatrixSpec value (real, integer, or boolean)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

Several MatrixSpecs do not apply to vectors (row, col, diagonal).

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Single*(), *MatrixSpecs_Block_Single*(), *MatrixSpecs_Col_Single*(), *MatrixSpecs_Diag_Single*(), *MatrixSpecs_Colwise_Single*(), *MatrixSpecs_Rowwise_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2
$specID = 3       ; ID 3 = mean (average)

 MsgBox ( 0, "Mean value of cells in", "row " & $rowindex & " = " & _
     _MatrixSpecs_Row_Single ( $matA, $rowindex, $specID ) )

_Eigen_CleanUp()
```



### MatrixSpecs_Rowwise_Single

Function Reference

# _Eigen_MatrixSpecs_Rowwise_Single

Evaluate and return a specified Matrix Spec for each row; store the result in a Colvector

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Rowwise_Single ( $matA, $specID[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $specID | ID (0-31) of the desired Matrix Spec to be returned |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

| | |
|---|---|
| Success: | array (32 x 2) containing all matrix specs (labels in column 0, values in column 1) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

This function does not accept any matrix inputs of, or related to, complex types.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, a Colvector with the same number of rows as input matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

Only a small subset of MatrixSpecs is supported here, viz.

| ID | MatrixSpec | Remarks |
|---|---|---|
| 1 | Sum | returns real |
| 2 | Product | returns real |
| 3 | Mean | returns real |
| 5 | Norm | returns real |
| 6 | Squared norm | returns real |
| 9 | Minimum value | returns real |
| 10 | Maximum value | returns real |

## Related

*MatrixSpecs*(), *MatrixSpecs_Single*(), *MatrixSpecs_Block_Single*(), *MatrixSpecs_Col_Single*(), *MatrixSpecs_Diag_Single*(), *MatrixSpecs_Row_Single*(), *MatrixSpecs_Colwise_Single*()

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$specID = 3       ; ID 3 = mean (average)

$matM = _MatrixSpecs_Rowwise_Single ( $matA, $specID )
_MatrixDisplay ( $matM, "Rowwise means" )

_Eigen_CleanUp()
```





# _Eigen_MatrixSpecs_Single

Evaluate and return one specifiied Matrix Spec for a matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_MatrixSpecs_Single ( $matA, $specID[, $epsilon = -1] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $specID | ID (0-31) of the desired Matrix Spec to be returned |
| $epsilon | **[optional]** precision of the comparison (default (-1): *1.0e-6* for float, *1.0e-12* for double) |

## Return Value

Success: the requested MatrixSpec value (real, integer, or boolean)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Precision parameter $epsilon applies only to the evaluation of the boolean specs.

All matrixSpecs are tabulated here.

## Related

*MatrixSpecs*(), *MatrixSpecs_Block_Single*(), *MatrixSpecs_Col_Single*(), *MatrixSpecs_Diag_Single*(), *MatrixSpecs_Row_Single*(), *MatrixSpecs_Colwise_Single*(), *MatrixSpecs_Rowwise_Single*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$specID = 3       ; ID 3 = mean (average)
MsgBox ( 0, "Mean value of cells in", "entire matrix = " & _
      _MatrixSpecs_Single ( $matA, $specID ))

_Eigen_CleanUp()
```





# _Eigen_Show_MatrixSpecs

Display the list of MatrixSpecs variables

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_MatrixSpecs ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True, displayed array |
| Failure: | n/a |

## Remarks

For demonstration purposes only (listing the variables); all values are set to missing (value: -1).

## Related

*Show_DecompSpecs(), Show_MatrixSpecs_Current()*

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

_Eigen_Show_MatrixSpecs()

_Eigen_CleanUp()
```





# _Eigen_Show_MatrixSpecs_Current

Display the current contents of MatrixSpecs

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_MatrixSpecs_Current ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True, displayed array |
| Failure: | n/a |

## Remarks

None.

## Related

*Show_DecompSpecs_Current*(), *Show_MatrixSpecs*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

_MatrixSpecs ( $matA )
_Eigen_Show_MatrixSpecs_Current()

_Eigen_CleanUp()
```

## Transformation

# Matrix Transformation

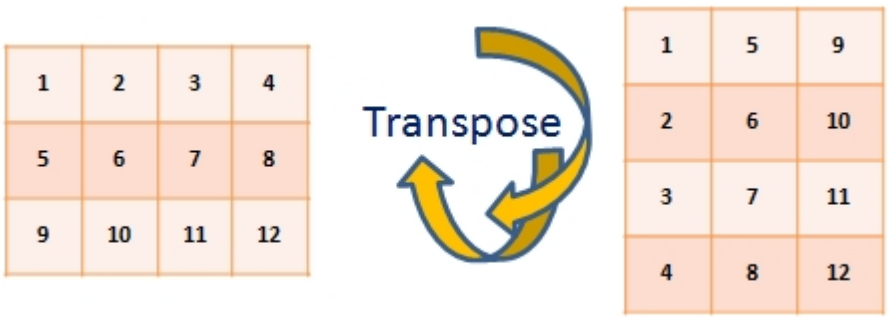
Four types of matrix transformation are available, each with an **_inPlace** counterpart:

| | |
|---|---|
| **_Eigen_Reverse** | exchange top-bottom and left-right (flip cross-wise) |
| **_Eigen_Transpose** | exchange rows and cols (flip along the diagonal) |
| **_Eigen_Adjoint** | ditto, but for complex matrices, equals **Tranpose** for real matrices |
| **_Eigen_Inverse** | (see below) |

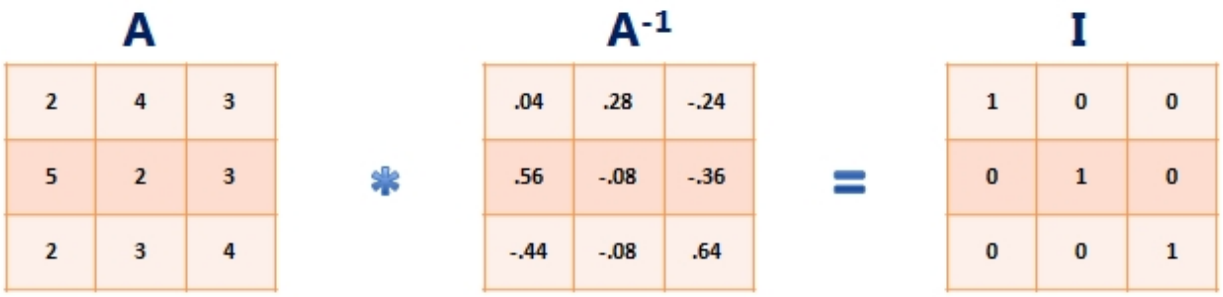
Reverse is simple: destination shape equals input, just the cell values are mirrored.


Unlike the other three transformations, Reverse can also be applied to matrix parts (block, column, diagonal, row).

Transpose (and Adjoint, the complex equivalent of transpose) are less obvious in execution; if you transpose a 3 x 4 input matrix, the result has its **dimensions flipped** (4 x 3), including all cell content. (NB See here for **ReDim**, a little-known alternative to transposition.)




And now it gets wild. Matrix division *does not exist!* The only thing we can say is that under certain strict conditions, an "inverse" matrix $\mathbf{A}^{-1}$ *may* exist that, when multiplied with the original input **A**, yields an Identity matrix (designated with capital **I**, the equivalent of a scalar ***1*** (unity)). Do not use **Eigen_Inverse** other than for tiny matrices (up to size ***4*** is often okay) to explore the concept. Anything larger requires the .inverse() member function of Eigen's heavy-duty decompositions, in which operational specs can be tightly controlled to deal with whatever quirky ill-conditioning your matrix may suffer from. You have been warned.






## Adjoint

Function Reference

# _Eigen_Adjoint

Complex equivalent of transposition (see _Eigen_Transpose)

```
#include <Eigen4AutoIt.au3>
_Eigen_Adjoint ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If a real (non-complex) matrix is supplied, the effect is the same as when calling _Eigen_Transpose().

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Adjoint_InPlace(), Transpose(), Transpose_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Adjoint ( $matA )
_MatrixDisplay ( $matR, "Adjoint A" )

_Eigen_CleanUp()
```





# _Eigen_Adjoint_InPlace

Complex equivalent of transposition (see _Eigen_Transpose); stored in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Adjoint_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix needs to be **square**.

The results are stored in the input matrix, replacing the original data.

## Related

*Adjoint*(), *Transpose*(), *Transpose_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"
```

```
_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Adjoint_InPlace ( $matA )
_MatrixDisplay ( $matA, "Adjoint A" )

_Eigen_CleanUp()
```





# _Eigen_Inverse

Compute the matrix inverse

```
#include <Eigen4AutoIt.au3>
_Eigen_Inverse ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

**A matrix inverse may not exist!**

Do not use this function for matrices of size 5 and larger; instead, use one of the main decompositions.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Inverse_InPlace(), PseudoInverse()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Inverse ( $matA )
_MatrixDisplay ( $matR, "Inverse A" )

_Eigen_CleanUp()
```



### Inverse_InPlace

Function Reference

# _Eigen_Inverse_InPlace

Compute the matrix inverse; stored in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Inverse_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

**A matrix inverse may not exist!**

The results are stored in the input matrix, replacing the original data.

## Related

*Inverse*(), *PseudoInverse*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Inverse_InPlace ( $matA )
_MatrixDisplay ( $matA, "Inverse A" )

_Eigen_CleanUp()
```



### PseudoInverse

Function Reference

# _Eigen_PseudoInverse

Compute the Penrose-Moore pseudo-inverse using Courrieu's algorithm

```
#include <Eigen4AutoIt.au3>
_Eigen_PseudoInverse ( $matA[, $matR = 0[, epsilon = -1 ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix [rowsA, colsA] |
| $matR | **[optional]** matrix ID of the results matrix (reversed dimensions: [**colsA**, **rowsA**]) |
| $epsilon | **[optional]** threshold to determine when Choleski diagonal values are to be considered non-zero |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The Penrose-Moore pseudo-inverse returns a matrix with **reversed dimensions** with respect to the input matrix.

If the input matrix **A** is square and a true inverse exists, this will be returned. However, if no true inverse exists (due to a null space), a pseudo-inverse will be computed that takes into account only those reprojected dimensions that have non-zero eigenvalues. This implies that the dimensions of a non-square

matrix **R** will be exchanged with respect to **A**, so matrix **R** has dimensions [colsA, rowsA]. Thus, unlike decompositions that return a true inverse and _Eigen_Inverse(), some form of defined inverse is *always* obtained, but caution is needed if the original input matrix is non-square.

Thsi function's underlying algorithm is adapted from Courrieu, P., Fast Computation of Moore-Penrose Inverse Matrices, *Neural Information Processing - Letters and Reviews* 8 (2), Aug 2005. A more traditional algorithm for computing a pseudo-inverse is provided by decomposition JacobiSVD. Courrieu's version as implemented here becomes increasingly more efficient than JacobiSVD's version for larger dimensions.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Inverse*(), *Inverse_InPlace*(), *JacobiSVD*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 3, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_PseudoInverse ( $matA )
_MatrixDisplay ( $matR, "Inverse A" )

_Eigen_CleanUp()
```



### Reverse

Function Reference

# _Eigen_Reverse

Mirror-exchange the cell contents of a matrix horizontally and vertically

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

| | |
|---|---|
| Success: | $matR |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Reverse ( $matA )
_MatrixDisplay ( $matR, "Reverse A" )

_Eigen_CleanUp()
```



### Reverse_Block

Function Reference

# _Eigen_Reverse_Block

Mirror-exchange a selected block's cell contents horizontally and vertically

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $startRow | row ID (base-0) of the topmost row of the block in matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in matrix A |
| $blockRows | block row dimensions |

| | |
|---|---|
| $blockCols | block column dimension |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2

$matR = _Eigen_Reverse_Block ( $matA, $startRow, $startCol, $blockRows,
$blockCols )
_MatrixDisplay ( $matR, "Reverse Block A" )

_Eigen_CleanUp()
```





# _Eigen_Reverse_Block_InPlace

Mirror-exchange a selected block's cell contents horizontally and vertically; stored in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Block_InPlace ( $matA, $startRow, $startCol, $blockRows,
$blockCols )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |
| $startRow | row ID (base-0) of the topmost row of the block in matrix A |
| $startCol | column ID (base-0) of the leftmost column of the block in matrix A |
| $blockRows | block row dimensions |
| $blockCols | block column dimension |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$startRow = 0     ; target block's top-left corner
$startCol = 0
$blockRows = 2    ; size of target block
$blockCols = 2

_Eigen_Reverse_Block_InPlace ( $matA, $startRow, $startCol, $blockRows,
$blockCols )
_MatrixDisplay ( $matA, "Reverse Block A" )

_Eigen_CleanUp()
```

Reverse_Col

Function Reference

# _Eigen_Reverse_Col

Mirror-exchange the cell contents of a selected column

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Col ( $matA, $colindex[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $colindex | column ID (base-0) of the column to act upon |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$colindex = 2

$matR = _Eigen_Reverse_Col ( $matA, $colindex )
_MatrixDisplay ( $matR, "Reverse Col A" )
```

```
_Eigen_CleanUp()
```





# _Eigen_Reverse_Col_InPlace

Mirror-exchange the cell contents of a selected column; stored in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Col_InPlace ( $matA, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $colindex | column ID (base-0) of the column to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$colindex = 2

_Eigen_Reverse_Col_InPlace ( $matA, $colindex )
```

```
_MatrixDisplay ( $matA, "Reverse Col A" )

_Eigen_CleanUp()
```





# _Eigen_Reverse_Diag

Mirror-exchange the cell contents of the diagonal

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Diag ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Reverse_Diag ( $matA )
```

```
_MatrixDisplay ( $matR, "Reverse Diag" )

_Eigen_CleanUp()
```





# _Eigen_Reverse_Diag_InPlace

Mirror-exchange the cell contents of the diagonal; stored in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Diag_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Reverse_Diag_InPlace ( $matA )
_MatrixDisplay ( $matA, "Reverse Diag" )

_Eigen_CleanUp()
```

# _Eigen_Reverse_InPlace

Mirror-exchange the cell contents of a matrix horizontally and vertically; stored in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the target matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Reverse*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Reverse_inPlace ( $matA )
_MatrixDisplay ( $matA, "Reverse A" )

_Eigen_CleanUp()
```

Reverse_Row

Function Reference

# _Eigen_Reverse_Row

Mirror-exchange the cell contents of a selected row

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Row ( $matA, $rowindex[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $rowindex | row ID (base-0) of the row to act upon |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*(), *Reverse_Row_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2

$matR = _Eigen_Reverse_Row ( $matA, $rowindex )
_MatrixDisplay ( $matR, "Reverse Row A" )
```

```
_Eigen_CleanUp()
```





# _Eigen_Reverse_Row_InPlace

Mirror-exchange the cell contents of a selected row; store in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Reverse_Row_InPlace ( $matA, $rowindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $rowindex | row ID (base-0) of the row to act upon |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Reverse*(), *Reverse_inPlace*(), *Reverse_Block*(), *Reverse_Col*(), *Reverse_Diag*(), *Reverse_Row*(), *Reverse_Block_InPlace*(), *Reverse_Col_InPlace*(), *Reverse_Diag_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2

_Eigen_Reverse_Row_InPlace ( $matA, $rowindex )
_MatrixDisplay ( $matA, "Reverse Row A" )
```

```
_Eigen_CleanUp()
```





# _Eigen_Transpose

Mirror-exchange all cell contents across the diagonal (flip dimensions)

```
#include <Eigen4AutoIt.au3>
_Eigen_Transpose ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the **reverse** dimensions of matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Adjoint()*, *Adjoint_InPlace()*, *Transpose_InPlace()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Transpose ( $matA )
_MatrixDisplay ( $matR, "Transpose A" )
```

```
_Eigen_CleanUp()
```





# _Eigen_Transpose_InPlace

Mirror-exchange cell contents across the diagonal (flip dimensions); stored in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Transpose_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The input matrix needs to be **square**.

If a real (non-complex) matrix is supplied, the effect is the same as when calling _Eigen_Transpose_InPlace().

The results are stored in the input matrix, replacing the original data.

## Related

*Adjoint*(), *Adjoint_InPlace*(), *Transpose*()

## Example

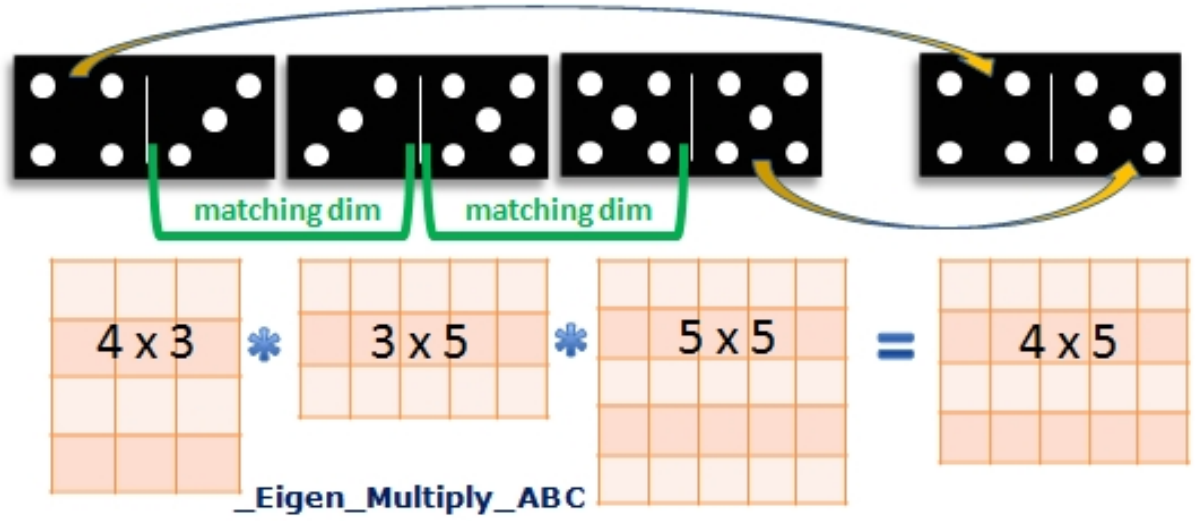
```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Transpose_InPlace ( $matA )
_MatrixDisplay ( $matA, "Transpose A" )
```

```
_Eigen_CleanUp()
```



## Multiplication

# Matrix Multiplication




Multiplying matrices is possible *only* ***if adjacent diminoes can be laid***, that is, *if*, for each pair of multiplicands, the number of cols in the first matrix matches the number of rows in the second. The final result will have as many rows as the first, and as many cols as the last matrix in the chain. This can cause extra headaches when attempting to matrix-multiply in-place (storing the final result in the first input matrix), or when multiplying with one or more transposed matrices.

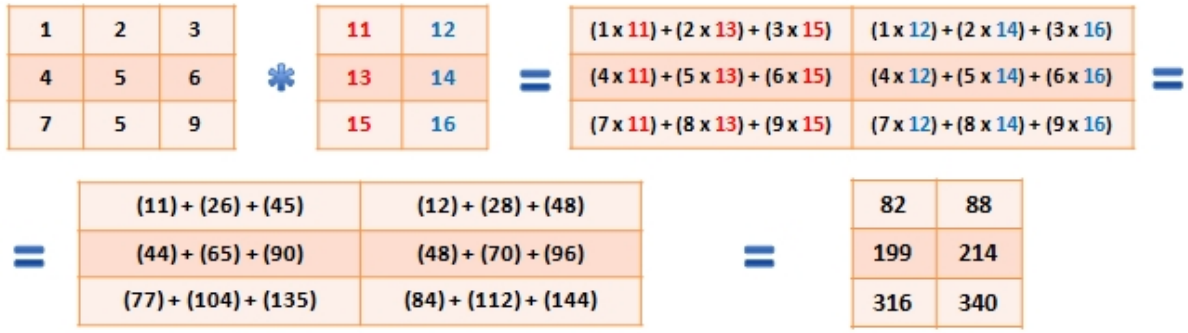
Some examples:

| | | |
|---|---|---|
| **_Eigen_Multiply_AA** | **_Eigen_Multiply_AAt** | **_Eigen_Multiply_AB_plusC** |
| **_Eigen_Multiply_AB** | **_Eigen_Multiply_AtA** | **_Eigen_Multiply_AB_minusC** |
| **_Eigen_Multiply_ABC** | **_Eigen_Multiply_AtAt** | **_Eigen_Multiply_A_BplusC** |
| **_Eigen_Multiply_ABCD** | **_Eigen_Multiply_AtBt** | **_Eigen_Multiply_A_BminusC** |

The third column contains functions with a second **Cwise** addition or subtraction, performed either *after* the multiplication (top two) or *before* (bottom two), with different(!) **C** dimensions.

Matrix multiplication differs from cellwise multiplication in that individual cell multiplications are **summed**. For each pair of matrices **AB** to be multiplied, each row of **A** and each column of **B** are traversed in turn, and their cell products added. Thus for the product to *exist*, the number of columns in **A** has to match the number of rows in **B**. Moreover, multiplication is not commutative; matrix product **AB** usually does not equal matrix product **BA**.

A numerical example may clarify this, here multiplying **A** (3x3) with **B** (3x2) to yield a (3x2) result. Each cell of the result then contains the sum of three pair-wise multiplications:




Three vector-specific products are also supported, namely: **_Eigen_DotProduct**, **_Eigen_CrossProduct**, and **_Eigen_OuterProduct** (the CrossProduct accepts length ***3*** vectors only; this is a current limitation of Eigen itself). Note that although the inputs are vectors, the first function returns a scalar, the second a vector, and the third a matrix.

| Function | Output | Remarks |
|---|---|---|
| _Eigen_DotProduct() | **scalar** | a.k.a. "scalar product", "inner product"<br>Algebraically, the sum of the products of two equal sequences of numbers; geometrically, the product of the two vectors' Euclidean magnitudes times the cosine of their angle.<br>Practical example: work done = dot product of force and displacement vectors |
| _Eigen_CrossProduct() | **vector** | a.k.a. "vector product"<br>Geometrically, the parallelogram area of the two vectors times the sine of their angle times a unit vector normal to the parallelogram's plane.<br>Practical examples: curl, Lorentz force, angular momentum, torque |
| _Eigen_OuterProduct() | **matrix** | a.k.a. "tensor product" (NB do not confuse with "exterior product")<br>Algebraically, creates a cellwise multiplcation table of a Rowvector times a Colvector<br>Practical examples: covariance, tensor of inertia |





# _Eigen_CrossProduct

Compute the cross-product of two same-sized vectors

```
#include <Eigen4AutoIt.au3>
_Eigen_CrossProduct ( $matA,[, $matB = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input vector |

| | |
|---|---|
| $matB | **[optional]** matrix ID of the second input **vector** |
| $matR | **[optional]** matrix ID of the results **vector** |

## Return Value

| | |
|---|---|
| Success: | $matR (vector) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

See the table of vector products for an overview.

The two input vectors have to have the same size of three elements (Eigen limitation). If vector **B** is not supplied, vector **A** is used twice.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, a Colvector of the same size as the input vectors) If results matrix **R** is not pre-supplied, it will be created.

## Related

*DotProduct(), OuterProduct()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 1, 3 )     ; vector
_MatrixDisplay ( $matA, "Rowvector A" )

$matB = _Eigen_CreateMatrix_Random ( 3, 1 )     ; vector
_MatrixDisplay ( $matB, "Colvector B" )

$matR = _Eigen_CrossProduct ( $matA, $matB )
_MatrixDisplay ( $matR, "Cross Product" )

_Eigen_CleanUp()
```





# _Eigen_DotProduct

Compute the dot-product of two same-sized vectors

```
#include <Eigen4AutoIt.au3>
_Eigen_DotProduct ( $matA,[, $matB = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input **vector** |
| $matB | **[optional]** matrix ID of the second input **vector** |

## Return Value

Success: dot product of the two vectors (scalar value)

Failure: False, and sets the @error flag to non-zero.

## Remarks

See the table of vector products for an overview.

The two input vectors have to be same-sized. If vector **B** is not supplied, vector **A** is used twice.

## Related

*CrossProduct*(), *OuterProduct*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 1, 3 )     ; vector
_MatrixDisplay ( $matA, "vector A" )

$matB = _Eigen_CreateMatrix_Random ( 1, 3 )     ; vector
_MatrixDisplay ( $matB, "vector B" )

MsgBox ( 0, "Dot Product", "A . B =  " & _Eigen_DotProduct ( $matA, $matB ) )

_Eigen_CleanUp()
```





# _Eigen_Multiply_A_BdivC

Multiply input matrix A with the result of Cwise-dividing matrix B by C

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_A_BdivC ( $matA, $matB, $matC[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
Matrices **B** and **C** have to have the same dimensions.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AB_plusC(), Multiply_AB_minusC(), Multiply_A_BminusC(),, Multiply_AB_divC()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Constant ( 3, 3, 3 )      ; 3rd param: value
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix_Constant ( 3, 4, 4 )      ; 3rd param: value
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix_Random ( 3, 4 )
_MatrixDisplay ( $matC, "matrix C" )

$matR = _Eigen_Multiply_A_BdivC ( $matA, $matB, $matC )
_MatrixDisplay ( $matR, "A.(B/C)" )

_Eigen_CleanUp()
```

# _Eigen_Multiply_A_BminusC

Multiply input matrix A with the result of Cwise-subtracting matrix C from matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_A_BminusC ( $matA, $matB, $matC[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
Matrices **B** and **C** have to have the same dimensions.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AB_plusC*(), *Multiply_AB_minusC*(), *Multiply_A_BplusC*(),, *Multiply_AB_divC*(), *Multiply_A_BdivC*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Constant ( 3, 3, 3 )      ; 3rd param: value
_MatrixDisplay ( $matA, "matrix A" )
```

```
$matB = _Eigen_CreateMatrix_Constant ( 3, 4, 4 )      ; 3rd param: value
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix_Ones ( 3, 4 )
_MatrixDisplay ( $matC, "matrix C" )

$matR = _Eigen_Multiply_A_BminusC ( $matA, $matB, $matC )
_MatrixDisplay ( $matR, "A.(B-C)" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_A_BplusC

Multiply input matrix A with the result of Cwise-adding matrices B and C

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_A_BplusC ( $matA, $matB, $matC[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
Matrices **B** and **C** have to have the same dimensions.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AB_plusC*(), *Multiply_AB_minusC*(), *Multiply_A_BminusC*(,, *Multiply_AB_divC*(), *Multiply_A_BdivC*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Constant ( 3, 3, 3 )      ; 3rd param: value
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix_Constant ( 3, 4, 4 )      ; 3rd param: value
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix_Ones ( 3, 4 )
_MatrixDisplay ( $matC, "matrix C" )

$matR = _Eigen_Multiply_A_BplusC ( $matA, $matB, $matC )
_MatrixDisplay ( $matR, "A.(B+C)" )

_Eigen_CleanUp()
```



### Multiply_AA

Function Reference

# _Eigen_Multiply_AA

Multiply input matrix A with itself

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AA ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrix **A** has to be **square**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as square matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA_InPlace()*, *Multiply_AB()*, *Multiply_ABC()*, *Multiply_ABCD()*, *Multiply_ABCDE()*, *Multiply_AAt()*, *Multiply_ABt()*, *Multiply_AtA()*, *Multiply_AtAt()*, *Multiply_AtB()*, *Multiply_AtBt()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )       ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Multiply_AA ( $matA )
_MatrixDisplay ( $matR, "A.A" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AA_InPlace

Multiply input matrix A with itself, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AA_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrix **A** has to be **square**.

The results are stored in the input matrix, replacing the original data.

## Related

*Multiply_AA*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )      ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Multiply_AA_InPlace ( $matA )
_MatrixDisplay ( $matA, "A.A" )

_Eigen_CleanUp()
```



### Multiply_AAt

Function Reference

# _Eigen_Multiply_AAt

Multiply input matrix A with itself transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AAt ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows and columns as the number of rows of matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt_InPlace*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )      ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Multiply_AAt ( $matA )
_MatrixDisplay ( $matR, "A.At" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AAt_InPlace

Multiply input matrix A with itself transposed, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AAt_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrix **A** has to be **square**.

The results are stored in the input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )       ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Multiply_AAt_InPlace ( $matA )
_MatrixDisplay ( $matA, "A.At" )

_Eigen_CleanUp()
```



### Multiply_AB

Function Reference

# _Eigen_Multiply_AB

Multiply input matrix A with input matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AB ( $matA, $matB[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB_InPlace*(), *Multiply_ABC*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matR = _Eigen_Multiply_AB ( $matA, $matB )
_MatrixDisplay ( $matR, "A.B" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AB_divC

Multiply input matrices A and B first, and then Cwise-divide by the contents of matrix C

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AB_divC ( $matA, $matB, $matC[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |

| | |
|---|---|
| $matC | matrix ID of the third input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
Matrix C has to have the same number of rows as matrix **A**, and the same number of columns as matrix **B**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AB_minusC(), Multiply_A_BplusC(), Multiply_A_BminusC(),, Multiply_A_BdivC()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Constant ( 3, 3, 3 )      ; 3rd param: value
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix_Constant ( 3, 4, 4 )      ; 3rd param: value
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix_Random ( 3, 4 )
_MatrixDisplay ( $matC, "matrix C" )

$matR = _Eigen_Multiply_AB_divC ( $matA, $matB, $matC )
_MatrixDisplay ( $matR, "(A.B)/C" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AB_InPlace

Multiply input matrix A with input matrix B, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AB_InPlace ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Both input matrices have to be **square** and same-sized.

The results are stored in the first input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "matrix B" )

_Eigen_Multiply_AB_InPlace ( $matA, $matB )
_MatrixDisplay ( $matA, "A.B" )

_Eigen_CleanUp()
```

# _Eigen_Multiply_AB_minusC

Multiply input matrices A and B first, and then Cwise-subtract the contents of matrix C

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AB_minusC ( $matA, $matB, $matC[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
Matrix **C** has to have the same number of rows as matrix **A**, and the same number of columns as matrix **B**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AB_plusC(), Multiply_A_BplusC(), Multiply_A_BminusC(),, Multiply_AB_divC(), Multiply_A_BdivC()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Constant ( 3, 3, 3 )      ; 3rd param: value
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix_Constant ( 3, 4, 4 )      ; 3rd param: value
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix_Ones ( 3, 4 )
_MatrixDisplay ( $matC, "matrix C" )

$matR = _Eigen_Multiply_AB_minusC ( $matA, $matB, $matC )
```

```
_MatrixDisplay ( $matR, "(A.B)-C" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AB_plusC

Multiply input matrices A and B first, and then Cwise-add the contents of matrix C

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AB_plusC ( $matA, $matB, $matC[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
Matrix C has to have the same number of rows as matrix **A**, and the same number of columns as matrix **B**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AB_minusC(), Multiply_A_BplusC(), Multiply_A_BminusC(),, Multiply_AB_divC(), Multiply_A_BdivC()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
$matA = _Eigen_CreateMatrix_Constant ( 3, 3, 3 )      ; 3rd param: value
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix_Constant ( 3, 4, 4 )      ; 3rd param: value
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix_Ones ( 3, 4 )
_MatrixDisplay ( $matC, "matrix C" )

$matR = _Eigen_Multiply_AB_plusC ( $matA, $matB, $matC )
_MatrixDisplay ( $matR, "(A.B)+C" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_ABC

Multiply input matrices A, B, and C

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABC ( $matA, $matB, $matC[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
The number of columns in matrix **B** has to match the number of rows in matrix **C**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **C**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matC, 1, 20 )
_MatrixDisplay ( $matC, "matrix C" )

$matR = _Eigen_Multiply_ABC ( $matA, $matB, $matC )
_MatrixDisplay ( $matR, "A.B.C" )

_Eigen_CleanUp()
```



### Multiply_ABC_InPlace

Function Reference

# _Eigen_Multiply_ABC_InPlace

Multiply input matrices A, B, and C, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABC_InPlace ( $matA, $matB, $matC )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**, and the number of columns in matrix **C**.
The number of columns in matrix **B** has to match the number of rows in matrix **C**.

The results are stored in the first input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matC, 1, 12 )
_MatrixDisplay ( $matC, "matrix C" )

_Eigen_Multiply_ABC_InPlace ( $matA, $matB, $matC )
_MatrixDisplay ( $matA, "A.B.C" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_ABCD

Multiply input matrices A, B, C, and D

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABCD ( $matA, $matB, $matC, $matD[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matD | matrix ID of the fourth input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
The number of columns in matrix **B** has to match the number of rows in matrix **C**.
The number of columns in matrix **C** has to match the number of rows in matrix **D**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **D**). If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt*(), *Multiply_ABCDE*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matC, 1, 20 )
_MatrixDisplay ( $matC, "matrix C" )
```

```
$matD = _Eigen_CreateMatrix ( 5, 6 )
_Eigen_SetLinSpaced_RowMajor ( $matD, 1, 30 )
_MatrixDisplay ( $matD, "matrix D" )

$matR = _Eigen_Multiply_ABCD ( $matA, $matB, $matC, $matD )
_MatrixDisplay ( $matR, "A.B.C.D" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_ABCD_InPlace

Multiply input matrices A, B, C, and D, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABCD_InPlace ( $matA, $matB, $matC, $matD )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matD | matrix ID of the fourth input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B** and the number of columns in matrix **D**.
The number of columns in matrix **B** has to match the number of rows in matrix **C**.
The number of columns in matrix **C** has to match the number of rows in matrix **D**.

The results are stored in the first input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matC, 1, 20 )
_MatrixDisplay ( $matC, "matrix C" )

$matD = _Eigen_CreateMatrix ( 5, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matD, 1, 15 )
_MatrixDisplay ( $matD, "matrix D" )

_Eigen_Multiply_ABCD_InPlace ( $matA, $matB, $matC, $matD )
_MatrixDisplay ( $matA, "A.B.C.D" )

_Eigen_CleanUp()
```



### Multiply_ABCDE

Function Reference

# _Eigen_Multiply_ABCDE

Multiply input matrices A, B, C, D, and E

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABCDE ( $matA, $matB, $matC, $matD, $matE[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matD | matrix ID of the fourth input matrix |
| $matE | matrix ID of the fifth input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

| | |
|---|---|
| Success: | $matR |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B**.
The number of columns in matrix **B** has to match the number of rows in matrix **C**.
The number of columns in matrix **C** has to match the number of rows in matrix **D**.
The number of columns in matrix **D** has to match the number of rows in matrix **E**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as input matrix **A**, and the same number of columns as input matrix **E**). If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD_InPlace*(),, *Multiply_ABCDE_InPlace*() *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matC, 1, 20 )
_MatrixDisplay ( $matC, "matrix C" )

$matD = _Eigen_CreateMatrix ( 5, 6 )
_Eigen_SetLinSpaced_RowMajor ( $matD, 1, 30 )
_MatrixDisplay ( $matD, "matrix D" )

$matE = _Eigen_CreateMatrix ( 6, 7 )
_Eigen_SetLinSpaced_RowMajor ( $matD, 1, 42 )
_MatrixDisplay ( $matD, "matrix D" )

$matR = _Eigen_Multiply_ABCDE ( $matA, $matB, $matC, $matD, %matE )
_MatrixDisplay ( $matR, "A.B.C.D.E" )

_Eigen_CleanUp()
```

Multiply_ABCDE_InPlace

Function Reference

# _Eigen_Multiply_ABCDE_InPlace

Multiply input matrices A, B, C, D, and E, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABCDE_InPlace ( $matA, $matB, $matC, $matD, $matE )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matC | matrix ID of the third input matrix |
| $matD | matrix ID of the fourth input matrix |
| $matE | matrix ID of the fifth input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of rows in matrix **B** and the number of columns in matrix **E**.
The number of columns in matrix **B** has to match the number of rows in matrix **C**.
The number of columns in matrix **C** has to match the number of rows in matrix **D**.
The number of columns in matrix **D** has to match the number of rows in matrix **E**.

The results are stored in the first input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
$matA = _Eigen_CreateMatrix ( 2, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 6 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matC = _Eigen_CreateMatrix ( 4, 5 )
_Eigen_SetLinSpaced_RowMajor ( $matC, 1, 20 )
_MatrixDisplay ( $matC, "matrix C" )

$matD = _Eigen_CreateMatrix ( 5, 6 )
_Eigen_SetLinSpaced_RowMajor ( $matD, 1, 30 )
_MatrixDisplay ( $matD, "matrix D" )

$matE = _Eigen_CreateMatrix ( 6, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matD, 1, 18 )
_MatrixDisplay ( $matD, "matrix D" )

_Eigen_Multiply_ABCDE_InPlace ( $matA, $matB, $matC, $matD, $matE )
_MatrixDisplay ( $matA, "A.B.C.D.E" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_ABt

Multiply input matrix A with input matrix B transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABt ( $matA, $matB, [, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of columns in matrix **A** has to match the number of columns in matrix **B**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as matrix **A**, and the same number of columns as the number of rows in matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt_InPlace*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matR = _Eigen_Multiply_ABt ( $matA, $matB )
_MatrixDisplay ( $matR, "A.Bt" )

_Eigen_CleanUp()
```



### Multiply_ABt_InPlace

Function Reference

# _Eigen_Multiply_ABt_InPlace

Multiply input matrix A with input matrix B transposed, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_ABt_InPlace ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Input matrices **A** and **B** have to be **square** and same-sized.

The results are stored in the first input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "matrix B" )

_Eigen_Multiply_ABt_InPlace ( $matA, $matB )
_MatrixDisplay ( $matA, "A.Bt" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AtA

Multiply input matrix A transposed with itself

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AtA ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows and columns as the number of columns of matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )      ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Multiply_AtA ( $matA )
_MatrixDisplay ( $matR, "At.A" )

_Eigen_CleanUp()
```



### Multiply_AtA_InPlace

Function Reference

# _Eigen_Multiply_AtA_InPlace

Multiply input matrix A transposed with itself, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
```

```
_Eigen_Multiply_AtA_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrix **A** has to be **square**.

The results are stored in the input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )      ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Multiply_AtA_InPlace ( $matA )
_MatrixDisplay ( $matA, "At.A" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AtAt

Multiply input matrix A transposed with itself transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AtAt ( $matA[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrix **A** has to be **square**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same dimensions as square matrix **A**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )      ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Multiply_AtAt ( $matA )
_MatrixDisplay ( $matR, "At.At" )

_Eigen_CleanUp()
```



### Multiply_AtAt_InPlace

Function Reference

# _Eigen_Multiply_AtAt_InPlace

Multiply input matrix A transposed with itself transposed, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AtAt_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the source matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrix **A** has to be **square**.

The results are stored in the input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )      ; square
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Multiply_AtAt_InPlace ( $matA )
_MatrixDisplay ( $matA, "At.At" )

_Eigen_CleanUp()
```



### Multiply_AtB

Function Reference

# _Eigen_Multiply_AtB

Multiply input matrix A transposed with input matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AtB ( $matA, $matB[, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of rows in matrix **A** has to match the number of rows in matrix **B**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as the number of columns of matrix **A**, and the same number of columns as the number of columns in matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB_InPlace*(), *Multiply_AtBt*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matR = _Eigen_Multiply_AtB ( $matA, $matB )
```

```
_MatrixDisplay ( $matR, "At.B" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AtB_InPlace

Multiply input matrix A transposed with input matrix B, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AtB_InPlace ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrices **A** and **B** have to be **square** and same-sized.

The results are stored in the input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace*(), *Multiply_AB_InPlace*(), *Multiply_ABC_InPlace*(), *Multiply_ABCD_InPlace*(), *Multiply_ABCDE_InPlace*(), *Multiply_AAt_InPlace*(), *Multiply_ABt_InPlace*(), *Multiply_AtA_InPlace*(), *Multiply_AtAt_InPlace*(), *Multiply_AtB*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "matrix B" )

_Eigen_Multiply_AtB_InPlace ( $matA, $matB )
_MatrixDisplay ( $matA, "At.B" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AtBt

Multiply input matrix A transposed with input matrix B transposed

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AtBt ( $matA, $matB, [, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

The number of rows in matrix **A** has to match the number of columns in matrix **B**.

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as the number of columns of matrix **A**, and the same number of columns as the number of rows of matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*Multiply_AA*(), *Multiply_AB*(), *Multiply_ABC*(), *Multiply_ABCD*(), *Multiply_ABCDE*(), *Multiply_AAt*(), *Multiply_ABt*(), *Multiply_AtA*(), *Multiply_AtAt*(), *Multiply_AtB*(), *Multiply_AtBt_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 3, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 12 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 3 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 12 )
_MatrixDisplay ( $matB, "matrix B" )

$matR = _Eigen_Multiply_AtBt ( $matA, $matB )
_MatrixDisplay ( $matR, "At.Bt" )

_Eigen_CleanUp()
```





# _Eigen_Multiply_AtBt_InPlace

Multiply input matrix A transposed with input matrix B transposed, and store the result in matrix A

```
#include <Eigen4AutoIt.au3>
_Eigen_Multiply_AtBt_InPlace ( $matA, $matB )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Input matrices **A** and **B** have to be **square** and same-sized.

The results are stored in the input matrix, replacing the original data.

## Related

*Multiply_AA_InPlace()*, *Multiply_AB_InPlace()*, *Multiply_ABC_InPlace()*, *Multiply_ABCD_InPlace()*, *Multiply_ABCDE_InPlace()*, *Multiply_AAt_InPlace()*, *Multiply_ABt_InPlace()*, *Multiply_AtA_InPlace()*, *Multiply_AtAt_InPlace()*, *Multiply_AtB_InPlace()*, *Multiply_AtBt()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matB, 2, 32 )
_MatrixDisplay ( $matB, "matrix B" )

_Eigen_Multiply_AtBt_InPlace ( $matA, $matB )
_MatrixDisplay ( $matA, "At.Bt" )

_Eigen_CleanUp()
```



### OuterProduct

Function Reference

# _Eigen_OuterProduct

Multiply a Rowvector with a Colvector to create a (multiplication table) matrix of their joint dimensions

```
#include <Eigen4AutoIt.au3>
_Eigen_OuterProduct ( $matA, $matB, [, $matR = 0] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input **vector** |
| $matB | matrix ID of the second input **vector** |
| $matR | **[optional]** matrix ID of the results **matrix** |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

See the table of vector products for an overview.

The two input vectors have to have the same size (>1); one should be a Rowvector; the other a Colvector (parsing order is unimportant here).

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, the same number of rows as the supplied Rowvector, and the same number of columns as the supplied Colvector) If results matrix **R** is not pre-supplied, it will be created.

## Related

*CrossProduct*(), *DotProduct*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 1, 5 )      ; row vector
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 5 )
_MatrixDisplay ( $matA, "Rowvector A" )

$matB = _Eigen_CreateMatrix ( 5, 1 )      ; col vector
_Eigen_SetLinSpaced_RowMajor ( $matB, 1, 5 )
_MatrixDisplay ( $matB, "Colvector B" )

$matR = _Eigen_OuterProduct ( $matA, $matB )
_MatrixDisplay ( $matR, "Outer Product" )

_Eigen_CleanUp()
```



### Decomposition

# Matrix Decomposition

Decompositions are the *Jaegers* of linear algebra, powerful number crunchers that can break down a single matrix into several others with desirable properties. These functions have operational flags to control how they approach the enemy, and you can specify exactly which outputs you wish to have returned; skipping some may substantially speed up computation. They provide inverse methods, different approaches to linear solving, and/or eigen analysis. **Eigen4AutoIt** supports most variants of the four main decompositions: **LowerUpper** ("LU"), **Householder** ("QR"), **Choleski**, and **Jacobi** singular value decomposition ("SVD," including Penrose-Moore pseudo-inverse), plus several helper decompositions (Hessenberg, RealSchur, ComplexScur, RealQZ, and Tridiagonalization). Please consult Eigen's documentation regarding their input limitations, as not all conditions are (or can be) tested beforehand.

Performing decompositions in **Eigen4AutoIt** is slightly more involved than calling the simpler functions,

because we usually wish to obtain only a few outputs out of many, and we don't want to parse an alphabet soup of zero pointers for everything we don't want. So instead, we use **_Eigen_SetActiveMatrix** with a string of only those generic matrix letters we actually need, e.g.:

```
_Eigen_SetActiveMatrix ( "EILUX", True )          ; True = clear the list of active matrices first
$specs = _Eigen_LowerUpper ( $matA, $enableSpecs, $fullPivoting, $matB )
$matI  =_Eigen_GetActiveMatrix ( "I" )   ; store newly-created output matrix's ID for inverse,
$matX =_Eigen_GetActiveMatrix ( "X" )  ; store new output matrix's ID for solution, etc...
```

Here we've enabled results matrices **E** (always the full decomposition), **I** (always the inverse), **L** & **U** (the main outputs of LowerUpper), and **X** (always the linear solver's result). Input matrices **A** (kernel, always present) and **B** (observations, required if fitting a model **X**) are still parsed normally, as are boolean flags $enableSpecs and $fullPivoting. The latter controls how **LowerUpper** works, the former determines whether an array of decomposition specs should be returned (here stored in variable $specs), containing a status code, the determinant, and/or various other data (a returned value of ***-1*** means that particular spec is unavailable for the selected decomposition). The same drill applies to **_Eigen_HouseholderQR**, **_Eigen_Choleski**, **_Eigen_JacobiSVD**, and, minus the specs array, to statistics functions **_Eigen_PCA** and **_Eigen_PCF** (with eight and four optional outputs respectively).

If you wish, you can alternatively provide any existing output container yourself (all optional matrix ID parameters are arranged alphabetically). In that case, you'll likely benefit from functions **_CreateOutput_FromDims** and **_CreateOutput_FromIDs**, which can predict the dimensions of most output matrices, given the ones for the input(s). In addition, you may wish to consult the provided tables that list the type and dimensions of each output for the main linear algebra and statistics procedures. It makes for riveting bedtime reading.

For details on the implications of using complex decompositions and other specifics regarding calling these functions in a complex work environment, see this outline and the table of real and complex inputs and outputs.





# Decomposition Outputs: Matrices

| Content | LU | QR | CHO | SVD | HESS | SCHU | QZ | LSQ | PCA |
|---|---|---|---|---|---|---|---|---|---|
| A | Kernel (input, always required) | | | | | | | | |
| B | Observations (input, optional, required for linear solver) | | | | | | data, required | Observations | **A**, centered |
| C | | | | | QR coeffs | | | | Covariance |
| D | | | Choleski diagonal | | | | | | **S**, proportional |
| E | full decomposition | | | | | | | | |
| H | | **H**coeffs | | | Hessenberg matrix | | | | |
| I | Inverse of A (may not exist) | | | | | | | | |
| L | (**L**)ower triangul. | | (**L**)ower triangul. | | | | | | |
| M | | | | | | | | | colwise Means **A** |
| P | Permutations | | | | Packed matrix | | | | **A**, (P)rojected |
| Q | permutations | matrix **Q** (unitary) | | | matrix **Q** (orthog.) | | matrix **Q** | | **B**, projected |
| R | **R**econstructed | matrix **R** up.trian. | **R**econstructed | | | | | | **R**econstructed |
| S | | | | eigen values | | | matrix **S** | | eigen values |
| T | | | | | | matrix **T** | matrix **T** | | |
| U | (**U**)pper triangul. | Thin **Q** | matrix **U** | eigen vectors | | matrix **U** | | | eigen vectors |
| V | | | | matrix **V** (unitary) | | | | | |
| W | weighting matrix (optional) | | | | | | | | |
| X | Linear solver's solution (**A** . **X** = **B**) | | | | | | | | |
| Z | | | | | | | matrix **Z** | | |

See the **Eigen** documentation (at http://eigen.tuxfamily.org/dox) and annotations in associated **AutoIt** wrapper and dll functions for more details regarding the various decomposition outputs.

Note that:
1. not all listed matrices are available for all variants of a particular decomposition;
2. not all internal matrices may be accessible through **Eigen**'s own member functions;
3. matrices **A**, **E**, **I**, **W**, and **X** always contain the same type of content.

Decomposition Outputs: Dimensions

# Decomposition Outputs: Dimensions

| Dims | LU | QR | CHO | SVD | HESS | SCHU | QZ | LSQ | PCA |
|---|---|---|---|---|---|---|---|---|---|
| A | rowsA colsA | rowsA colsA | rowsA colsA | rowsA colsA | rowsA colsA | rowsA[1] colsA | rowsA colsA | rowsA colsA | rowsA colsA |
| B | rowsA colsB | rowsA[2] 1 | rowsA colsB | rowsA colsB | | | rowsA colsA | rowsA colsB | rowsA colsA |
| C | | | | | coeffs[3] 1 | | | | colsA colsA |
| D | | | rowsA[4] 1 | | | | | | 1 colsA |
| E | rowsA colsA | rowsA colsA | rowsA colsA | | | | | | |
| H | | diagA 1 | | | rowsA colsA | rowsA[1] colsA | | | |
| I | rowsA colsA | colsA colsA | | colsA[5] rowsA | | | | | |
| L | rowsA rowsA | | rowsA colsA | | | | | | |
| M | | | | | | | | | 1 colsA |
| P | rowsA rowsA | | | | rowsA colsA | | | | colsA rowsA |
| Q | colsA colsA | rowsA rowsA | | | rowsA colsA | | rowsA colsA | | colsA rowsA |
| R | rowsA colsA | diagA diagA | rowsA colsA | | | | | | rowsA colsA |
| S | | | | rowsA colsA | | | rowsA colsA | | 1 colsA |
| T | | | | | | rowsA colsA | rowsA colsA | | |
| U | rowsA colsA | rowsA diagA | rowsA colsA | rowsA[6] rowsA | | rowsA colsA | | | colsA colsA |
| V | | | | colsA[6] colsA | | | | | |
| W | | | | | | | | rowsA rowsA | |
| X | colsA colsB | colsA[2] 1 | colsA colsB | colsA colsB | | | | colsA colsB | |
| Z | | | | | | | rowsA colsA | | |

**diagA** = min(rowsA,colsA)
**coeffs** = ( ( (rowsA*rowsA) - rowsA) /2) - (rowsA - 1)

[1] : RealSchur takes a Hessenberg input matrix **H** instead of matrix **A**, if $computeFromHessenberg =

True
[2]: Eigen's QR solver can handle observation vectors only (for now), no matrices yet
[3]: vector of Householder coefficients
[4]: Cholesky diagonal vector, available only if $robust = True
[5]: pseudo-inverse, maps the relevant part of **X** only; remaining rows of **B** map the null space of **S** (irrelevant)
[6]: if $computeThin = True, dimensions of matrices **U** and **V** depend on whether rowsA >? colsA (see script)





# _Eigen_Decomp_Choleski

Perform a Choleski decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_Choleski ( $matA[, $enableSpecs = False[, $robust = False[,
$matB = 0[, $matD = 0[, $matE = 0[, $matL = 0[, $matR = 0[, $matU = 0[,
$matX = 0 ]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], **square**) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $robust | **[optional]** **True**: use LLDT version; **False**: use LLT version |
| $matB | **[optional]** matrix ID of the observations matrix for linear model fit ( **A . X** = **B** ) [rowsA, colsB] |
| $matD | **[optional]** matrix ID of the Choleski diagonal [rowsA, 1] (robust version only) |
| $matE | **[optional]** matrix ID of the full decomposition [rowsA, rowsA] |
| $matL | **[optional]** matrix ID of the lower diagonal part [rowsA, rowsA] |
| $matR | **[optional]** matrix ID of the reconstructed matrix [rowsA, rowsA] |
| $matU | **[optional]** matrix ID of the upper diagonal part [rowsA, rowsA] |
| $matX | **[optional]** matrix ID of the linear model fit ( **A . X** = **B** ) |

## Return Value

| | |
|---|---|
| Success: | True, or (if $enableSpecs = **True**) array $decompSpecs |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on each individual decomposition. See also Eigen's Linear Algebra Tutorial. More specific documentation on this specific decomposition can be found here (robust LLDT version) and here (regular LLT version)

Please do not use Choleski decomposition to determine *whether* a system of equations has a solution.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

This function does not accept any matrix inputs of, or related to, complex types.

Setting $robust = True deploys the LLDT version of this decomposition, which requires a **square**, positive or negative semi-definite input matrix.
Setting $robust = False deploys the regular LLT version of this decomposition, which is more restrictive than the robust version in requiring a **square**, positive definite input matrix (it is unstable otherwise).

The number of rows of optional matrix **B** (used to fit a linear model) has to match the number of rows of matrix **A**.

Choleski diagonal matrix **D** is a Colvector with the same number of rows as the diagonal of input matrix **A**.

Linear model matrix **X** is available only if input matrix **B** is provided.

Choleski decomposition can in principle act on either triangular, and Eigen itself is able to switch (default: Lower). However, **Eigen4AutoIt** is restricted to acting on the lower triangular only (no such switch is available here).

## Related

*Decomp_LowerUpper(), Decomp_HouseholderQR(), Decomp_JacobiSVD()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "ELRU", True )   ; True = reset all flags first

_Eigen_Decomp_Choleski ( $matA )    ; the decomposition call
If @error Then Exit

$matE = _Eigen_GetActiveMatrix ( "E" )    ; collect new IDs one at a time
$matL = _Eigen_GetActiveMatrix ( "L" )
$matR = _Eigen_GetActiveMatrix ( "R" )
$matU = _Eigen_GetActiveMatrix ( "U" )
```

```
_MatrixDisplay ( $matE, "Choleski E matrix" )
_MatrixDisplay ( $matL, "Choleski L matrix" )
_MatrixDisplay ( $matR, "Choleski R matrix" )
_MatrixDisplay ( $matU, "Choleski U matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_Decomp_ComplexSchur

Perform a ComplexSchur helper decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_ComplexSchur ( $matA[, $enableSpecs = False[, $computeU =
True[, $computeFromHessenberg = False[, $matH = 0[, $matQ = 0[, $matT =
0[, $matU = 0[, $maxIter = -1 ]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $computeU | **[optional]** **True**: generate output $matU; **False**: don't |
| $computeFromHessenberg | **[optional]** **True**: use $matH instead of $matA; **False**: don't |
| $matH | **[optional]** matrix ID of the optional alternative Hessenberg input matrix [rowsA, rowsA] |
| $matQ | **[optional]** matrix ID of the Hessenberg orthogonal transformation matrix [rowsA, rowsA] |
| $matT | **[optional]** matrix ID of the quasi-triangular matrix [rowsA, rowsA] |
| $matU | **[optional]** matrix ID of the orthogonal matrix [rowsA, rowsA] |
| $maxIter | **[optional]** maximum number of iterations for achieving convergence |

## Return Value

Success: True, or (if $enableSpecs = **True** or $maxiter > 0) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of eigensolvers for a general overview and links to extensive documentation on each individual solver. More specific documentation on this specific function can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

This function expects a complex input matrix; all outputs are likewise always complex.

This decomposition normally requires a **square** input matrix **A**. Alternatively, it can take a (square) Hessenberg matrix **H** ($matH) as input, if the appropriate flag is set (i.e., $computeFromHessenberg = True). If so, it also references the associated orthogonal transformation matrix **Q**, which should then be provided directly as parsed matrix ID for $matQ. If a Hessenberg input matrix is selected *without* provding matrix **Q**, an Identity matrix will be deployed in its stead.

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Eigen's default maximum number of iterations to achieve convergence is 40 times rowsA; this particular function alternatively accepts a user-defined maximum parsed in parameter $maxiter.

## Related

*Decomp_Hessenberg*(), *Decomp_RealQZ*(), *Decomp_RealSchur*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp( "complexfloat" )

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "TU", True )     ; True = reset all flags first

_Eigen_Decomp_ComplexSchur ( $matA )      ; the decomposition call
If @error Then Exit

$matT = _Eigen_GetActiveMatrix ( "T" )    ; collect new IDs one at a time
$matU = _Eigen_GetActiveMatrix ( "U" )

_MatrixDisplay_Real ( $matT, "ComplexSchur T matrix, real" )
_MatrixDisplay_Imag ( $matT, "ComplexSchur T matrix, imag" )

_MatrixDisplay_Real ( $matU, "ComplexSchur U matrix, real" )
_MatrixDisplay_Imag ( $matU, "ComplexSchur U matrix, imag" )

_Eigen_CleanUp()
```



## Decomp_Hessenberg

Function Reference

# _Eigen_Decomp_Hessenberg

Perform a Hessenberg helper decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_Hessenberg ( $matA[, $enableSpecs = False[, $matC = 0[,
$matH = 0[, $matP = 0[, $matQ = 0 ]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $matC | **[optional]** matrix ID of the HouseholderQR coefficients (dimensions: see Remarks) |
| $matH | **[optional]** matrix ID of the Hessenberg matrix [rowsA, rowsA] |
| $matP | **[optional]** matrix ID of the Packed matrix [rowsA, rowsA] |
| $matQ | **[optional]** matrix ID of the orthogonal transformation matrix [rowsA, rowsA] |

## Return Value

Success: True, or (if $enableSpecs = **True**) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on each individual decomposition. More specific documentation on this specific decomposition can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

This decomposition requires a real or complex **square** input matrix **A**. The matrix type of all outputs reflects the input matrix's type.

The HouseholderQR coefficients matrix $matC is a Colvector of diagonal size -1.

Hessenberg and Tridiagonlization are related; both accept either a real or a complex input matrix, several optional outputs are available in both decompositions, and their type depends on the input matrix type (see table). Note that Tridiagonalization always returns matrices **D**, **T**, and **S** as type real, whereas the matrix type of **C**, **P**, and **Q** (as in the Hessenberg case) reflect the input type.

| Function | Variants | Inputs | Real Outputs | Complex Outputs |
|---|---|---|---|---|
| Hessenberg | real | real (**A**) | **C**: Householder coeffs<br>**H**: Hessenberg matrix<br>**P**: packed matrix | |

| | | | | |
|---|---|---|---|---|
| | | | **Q**: orthogonal matrix | |
| | complex | complex (**A**) | | **C**: Householder coeffs<br>**H**: Hessenberg matrix<br>**P**: packed matrix<br>**Q**: orthogonal matrix |
| Tridiagonalization | real | real (**A**) | **C**: Householder coeffs<br>**D**: diagonal<br>**P**: packed matrix<br>**Q**: orthogonal matrix<br>**S**: subDiagonal<br>**T**: triDiagonal | |
| | complex | complex (**A**) | **D**: diagonal<br>**T**: triDiagonal<br>**S**: subDiagonal | **C**: Householder coeffs<br>**P**: packed matrix<br>**Q**: unitary matrix |

## Related

*Decomp_RealQZ*(), *Decomp_RealSchur*(), *Decomp_Tridiagonalization*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "HQ", True )     ; True = reset all flags first

_Eigen_Decomp_Hessenberg ( $matA )        ; the decomposition call
If @error Then Exit

$matH = _Eigen_GetActiveMatrix ( "H" )    ; collect new IDs one at a time
$matQ = _Eigen_GetActiveMatrix ( "Q" )

_MatrixDisplay ( $matH, "Hessenberg H matrix" )
_MatrixDisplay ( $matQ, "Hessenberg Q matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_Decomp_HouseholderQR

Perform a HouseholderQR decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_HouseholderQR ( $matA[, $enableSpecs = False[, $fullPivoting
= True[, $colPivoting = False[, $computeThin = False[, $matB = 0[, $matE =
0[, $matH = 0[, $matI = 0[, $matQ = 0[, $matR = 0[, $matU = 0[, $matX =
0[, $epsilon = -1 ]]]]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, colsA]) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $fullPivoting | **[optional]** See Table in Remarks |
| $colPivoting | **[optional]** See Table in Remarks |
| $computeThin | **[optional]** **True**: generate the "Thin" version of matrix **Q** in output **U**; **False**: don't |
| $matB | **[optional]** matrix ID of the observations matrix for linear model fit ( **A . X** = **B** ) [rowsA, **1**] |
| $matE | **[optional]** matrix ID of the full QR decomposition matrix [rowsA, colsA] |
| $matH | **[optional]** matrix ID of the H coefficients matrix (dimensions: See Remarks) |
| $matI | **[optional]** matrix ID of the inverse of A (may not exist) [**colsA**, colsA] |
| $matQ | **[optional]** matrix ID of the first part of the decomposition [rowsA, rowsA] |
| $matR | **[optional]** matrix ID of the second part of the decomposition (dimensions: See below) |
| $matU | **[optional]** matrix ID of the upper-triangular matrix (dimensions: See Remarks) |
| $matX | **[optional]** matrix ID of the linear model fit ( **A . X** = **B** ) |
| $epsilon | **[optional]** threshold to determine when pivots are to be considered non-zero |

## Return Value

Success: True, or (if $enableSpecs = **True**) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on each individual decomposition. See also Eigen's Linear Algebra Tutorial. More specific documentation on this specific decomposition can be found here (full pivoting), here (column pivoting) and here (no pivoting).

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

Flags $FullPivoting and $ColPivoting jointly control how HouseholderQR operates:

| $FullPivoting | $ColPivoting | Action | Unavailable outputs | Speed | Accuracy |
|---|---|---|---|---|---|
| **False** | **False** | No Pivoting | matrix **I**, matrix **R** | ++ | + |

| | | | | | |
|---|---|---|---|---|---|
| **True** | **False** | Full Pivoting | matrix **R** | - | +++ |
| **False** | **True** | Column Pivoting | none | + | ++ |
| **True** | **True** | Full Pivoting | matrix **R** | - | +++ |

If $FullPivoting = False, input matrix **A** must be invertible; it is the user's responsibility to ensure this. Otherwise call with $FullPivoting = True (slower). Moreover if no pivoting is selected, any user-defined $epsilon will be ignored.

The number of rows of optional input matrix **B** (used to fit a linear model) has to match the number of rows of matrix **A**. Furthermore, matrix **B** has to be a Colvector here, as Eigen's QR solver cannot handle observation matrices yet, only vectors.

Householder coefficients matrix **H** is a Colvector with the same number of rows as the diagonal of input matrix **A**.

The inverse matrix **I** is available only if some form of pivoting is enabled. Note that for a non-square input matrix, this reduces the dimensions of **I** to a square matrix of size colsA. If the input matrix **A** is not invertible, the returned matrix **I** has undefined coefficients.

Decomposition matrix **R** is square, with the same number of rows and columns as the diagonal of input matrix **A**. It is available only in the column-Pivoting version of this decomposition.

Upper triangular matrix **U** is the "Thin" version of matrix **Q**, enabled only if $computeThin = True. It has the same number of rows as matrix **A**, and the same number of columns as the diagonal of input matrix **A**. Thin-**Q** (matrix **U**) will only have fewer columns than **Q** if rowsA is smaller than colsA; otherwise it just duplicates **Q**.

Linear model matrix **X** is available only if input matrix **B** is provided.

## Related

*Decomp_Choleski*(), *Decomp_LowerUpper*(), *Decomp_JacobiSVD*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "EHIQ", True )   ; True = reset all flags first

_Eigen_Decomp_HouseholderQR ( $matA )     ; the decomposition call
If @error Then Exit

$matE = _Eigen_GetActiveMatrix ( "E" )    ; collect new IDs one at a time
$matH = _Eigen_GetActiveMatrix ( "H" )
$matI = _Eigen_GetActiveMatrix ( "I" )
$matQ = _Eigen_GetActiveMatrix ( "Q" )

_MatrixDisplay ( $matE, "Householder E matrix" )
_MatrixDisplay ( $matH, "Householder H matrix" )
_MatrixDisplay ( $matI, "Householder I matrix" )
```

```
_MatrixDisplay ( $matQ, "Householder Q matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_Decomp_JacobiSVD

Perform a JacobiSVD decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_JacobiSVD ( $matA[, $enableSpecs = False[, $fullPivoting =
False[, $colPivoting = True[, $computeThin = False[, $eigenvaluesAsVector
= False[, $matB = 0[, $matI = 0[, $matS = 0[, $matU = 0[, $matV = 0[,
$matX = 0[, $epsilon = -1 ]]]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, colsA]) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $fullPivoting | **[optional]** See Table in Remarks |
| $colPivoting | **[optional]** See Table in Remarks |
| $computeThin | **[optional]** See Table in Remarks |
| $eigenvaluesAsVector | **[optional]** **True**: store **S** as Colvector; **False**: store **S** as diagonal of square matrix |
| $matB | **[optional]** matrix ID of the observations matrix for linear model fit ( **A . X** = **B** ) [rowsA, colsB] |
| $matI | **[optional]** matrix ID of the **pseudo**-inverse of **A** [**colsA**, **rowsA**] |
| $matS | **[optional]** matrix ID of the eigenvalues [rowsA, colsA] |
| $matU | **[optional]** matrix ID of the eigenvectors (dimensions: see Remarks) |
| $matV | **[optional]** matrix ID of the right-singular vectors (dimensions: see Remarks) |
| $matX | **[optional]** matrix ID of the linear model fit ( **A . X** = **B** ) |
| $epsilon | **[optional]** threshold below which eigenvalues are set to zero |

## Return Value

Success: True, or (if $enableSpecs = **True**) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on each individual decomposition. See also Eigen's Linear Algebra Tutorial. More specific documentation on this specific decomposition can be found here.

JacobiSVD is an extremely powerful and stable (but relatively slow) decomposition, forming the core of Principal Components Analysis, explained in detail in **Eigen4AutoIt**'s PCA Tutorial.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

Technically, JacobiSVD requires a square input matrix. For non-square matrices, a HouseholderQR preConditioner is therefore applied first (slower). Flags $FullPivoting and $ColPivoting jointly control how this Preconditioner operates:

| $FullPivoting | $ColPivoting | Action | Speed | Accuracy |
|---|---|---|---|---|
| **False** | **False** | No Preconditioner | ++ | + |
| **True** | **False** | Full Pivoting Preconditioner | - | +++ |
| **False** | **True** | Column Pivoting Preconditioner | + | ++ |
| **True** | **True** | Full Pivoting Preconditioner | - | +++ |

If $FullPivoting = False, input matrix **A** must be invertible; it is the user's responsibility to ensure this. Otherwise call with $FullPivoting = True (slower). The $computeThin option is unavailable if using full-pivoting preconditioning.

$epsilon provides a lower bound on acceptable eigenvalues; it offers a way to filter out the least significant dimensional contributions to overall variability. This can stabilise the result, but at the cost of some (small) loss of information. If $epsilon is user-defined, it should be tiny, but larger than the current level of precision (which is used by default, if $epsilon = **-1**). It **cannot** be zero; if zero is parsed, it will be ignored.

The number of rows of optional matrix **B** (used to fit a linear model) has to match the number of rows of matrix **A**.

JacobiSVD is special in providing the Penrose-Moore **pseudo-inverse** as optional matrix output **I**. If the input matrix **A** is square and a true inverse exists, this will be returned. However, if no true inverse exists (due to a null space), a pseudo-inverse will be computed that takes into account only those reprojected dimensions that have non-zero eigenvalues, thus mapping only the *relevant part* of matrix **X** (as the remaining rows of **B** would map the null space of **S**, and are therefore irrelevant). This implies that the dimensions of a non-square matrix **I** will be exchanged with respect to **A**, so matrix **I** has dimensions [colsA, rowsA]. Thus, unlike all other decompositions and _Eigen_Inverse(), some form of defined inverse is *always* obtained, but caution is needed if the original input matrix is non-square. An alternative, generally faster algorithm to obtain the pseudo-inverse is provided by _Eigen_PseudoInverse().

Given an input matrix **A** with dimensions [rowsA, colsA], the $computeThin flag controls the dimensions of output matrices **U** and **V** as follows:

| $computeThin | Shape of input matrix A | left-singular matrix U | right-singular matrix V |
|---|---|---|---|
| **False** | **Any** | [ rowsA, rowsA ] | [ colsA, colsA ] |
| **True** | **rowsA <= colsA** | [ rowsA, rowsA ] | [ colsA, rowsA ] |
| | **rowsA > colsA** | [ rowsA, colsA ] | [ colsA, colsA ] |

Linear model matrix **X** is available only if input matrix **B** is provided.

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Computation is faster if the input matrix is square (no QR PreConditioner applied), and/or if output matrices **U** and **V** are not to be generated.

Even if a complex input matrix is supplied, returned eigenvalues remain real only, stored in the real part of generic output matrix **S**. The latter's shape can be controlled through boolean flag $eigenvaluesAsVector; if True, a Colvector is returned. By default, **S** is returned as a diagonal matrix of matrix **A**'s dimensions to facilitate subsequent multiplication.

## Related

*Decomp_Choleski*(), *Decomp_LowerUpper*(), *Decomp_HouseholderQR*(), *PCA*(), *PseudoInverse*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "ISUV", True )   ; True = reset all flags first

_Eigen_Decomp_JacobiSVD ( $matA )   ; the decomposition call
If @error Then Exit

$matI = _Eigen_GetActiveMatrix ( "I" )    ; collect new IDs one at a time
$matS = _Eigen_GetActiveMatrix ( "S" )
$matU = _Eigen_GetActiveMatrix ( "U" )
$matV = _Eigen_GetActiveMatrix ( "V" )

_MatrixDisplay ( $matI, "Jacobi I matrix" )
_MatrixDisplay ( $matS, "Jacobi S matrix" )
_MatrixDisplay ( $matU, "Jacobi U matrix" )
_MatrixDisplay ( $matV, "Jacobi V matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_Decomp_LowerUpper

Perform a LowerUpper (LU) decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_LowerUpper ( $matA[, $enableSpecs = False[, $fullPivoting =
True[, $matB = 0[, $matE = 0[, $matI = 0[, $matL = 0[, $matP = 0[, $matQ =
0[, $matR = 0[, $matU = 0[, $matX = 0[, $epsilon = -1 ]]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, colsA]) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $fullPivoting | **[optional]** **True**: enable Full Pivoting; **False**: use partial Pivoting (potentially **unstable**) |
| $matB | **[optional]** matrix ID of the observations matrix for linear model fit ( **A . X** = **B** ) [rowsA, colsB] |
| $matE | **[optional]** matrix ID of the full decomposition [rowsA, colsA] |
| $matI | **[optional]** matrix ID of the inverse of **A** (may not exist) [rowsA, colsA] |
| $matL | **[optional]** matrix ID of the Lower part of the decomposition [rowsA, rowsA] |
| $matP | **[optional]** matrix ID of the first permutations matrix [rowsA, rowsA] |
| $matQ | **[optional]** matrix ID of the second permutations matrix [colsA, colsA] |
| $matR | **[optional]** matrix ID of the reconstructed matrix [rowsA, colsA] |
| $matU | **[optional]** matrix ID of the Upper part of the decomposition [rowsA, colsA] |
| $matX | **[optional]** matrix ID of the linear model fit ( **A . X** = **B** ) |
| $epsilon | **[optional]** threshold to determine when pivots are to be considered non-zero |

## Return Value

Success: True, or (if $enableSpecs = **True**) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on each individual decomposition. See also Eigen's Linear Algebra Tutorial. More specific documentation on this specific decomposition can be found here (full pivoting version) and here (partial pivoting version).

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

Performing this decomposition with partial pivoting (that is, setting $FullPivoting = False) is considered numerically stable *only* for square, invertible matrices; it is the user's responsibility to ensure this. Otherwise call with $FullPivoting = True (the default, slower). Moreover, if partial pivoting is selected, any user-defined $epsilon will be ignored.

The number of rows of optional input matrix **B** (used to fit a linear model) has to match the number of rows of matrix **A**.

Inverse matrix **I** is available only if input matrix **A** is square.

Linear model matrix **X** is available only if input matrix **B** is provided.

## Related

*Decomp_Choleski()*, *Decomp_HouseholderQR()*, *Decomp_JacobiSVD()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "ELPQRU", True ) ; True = reset all flags first

_Eigen_Decomp_LowerUpper ( $matA )  ; the decomposition call
If @error Then Exit

$matE = _Eigen_GetActiveMatrix ( "E" )    ; collect new IDs one at a time
$matL = _Eigen_GetActiveMatrix ( "L" )
$matP = _Eigen_GetActiveMatrix ( "P" )
$matQ = _Eigen_GetActiveMatrix ( "Q" )
$matR = _Eigen_GetActiveMatrix ( "R" )
$matU = _Eigen_GetActiveMatrix ( "U" )

_MatrixDisplay ( $matE, "LowerUpper E matrix" )
_MatrixDisplay ( $matL, "LowerUpper L matrix" )
_MatrixDisplay ( $matP, "LowerUpper P matrix" )
_MatrixDisplay ( $matQ, "LowerUpper Q matrix" )
_MatrixDisplay ( $matR, "LowerUpper R matrix" )
_MatrixDisplay ( $matU, "LowerUpper U matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_Decomp_RealQZ

Perform a RealQZ helper decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_RealQZ ( $matA, $matB[, $enableSpecs = False[, $matQ = 0[,
```

```
$matS = 0[, $matT = 0[, $matZ = 0[, $maxIter = -1 ]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix (dimensions: [rowsA, rowsA], square) |
| $matB | matrix ID of the second input matrix [rowsA, rowsA] |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $matQ | **[optional]** matrix ID of the first orthogonal matrix [rowsA, rowsA]) |
| $matS | **[optional]** matrix ID of the upper quasi-triangular matrix [rowsA, rowsA]) |
| $matT | **[optional]** matrix ID of the upper-triangular matrix [rowsA, rowsA]) |
| $matZ | **[optional]** matrix ID of the second orthogonal matrix [rowsA, rowsA]) |
| $maxIter | **[optional]** maximum number of iterations for achieving convergence |

## Return Value

Success: True, or (if $enableSpecs = **True** or $maxiter > 0) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on each individual decomposition. More specific documentation on this specific decomposition can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

This function does not accept any matrix inputs of, or related to, complex types.

This decomposition requires **two** square, same-sized real input matrices ($matA and $matB).

Computation is faster if output matrices **Q** and **Z** are not to be generated.

## Related

*Decomp_Hessenberg(), Decomp_RealSchur()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )
```

```
$matB = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matB, "matrix B" )

_Eigen_SetActiveMatrix ( "QSTZ", True )   ; True = reset all flags first

_Eigen_Decomp_RealQZ ( $matA, $matB )     ; the decomposition call
If @error Then Exit

$matQ = _Eigen_GetActiveMatrix ( "Q" )    ; collect new IDs one at a time
$matZ = _Eigen_GetActiveMatrix ( "Z" )

_MatrixDisplay ( $matQ, "RealQZ Q matrix" )
_MatrixDisplay ( $matZ, "RealQZ Z matrix" )

; instead of storing matrix IDs, they can also be parsed directly
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "S" ), "RealQZ S matrix" )
_MatrixDisplay ( _Eigen_GetActiveMatrix ( "T" ), "RealQZ T matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```





# _Eigen_Decomp_RealSchur

Perform a RealSchur helper decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_RealSchur ( $matA[, $enableSpecs = False[, $computeU =
True[, $computeFromHessenberg = False[, $matH = 0[, $matQ = 0[, $matT =
0[, $matU = 0[, $maxIter = -1 ]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $computeU | **[optional]** **True**: generate output $matU; **False**: don't |
| $computeFromHessenberg | **[optional]** **True**: use $matH instead of $matA; **False**: don't |
| $matH | **[optional]** matrix ID of the optional alternative Hessenberg input matrix [rowsA, rowsA] |
| $matQ | **[optional]** matrix ID of the Hessenberg orthogonal transformation matrix [rowsA, rowsA] |
| $matT | **[optional]** matrix ID of the quasi-triangular matrix [rowsA, rowsA] |

| | |
|---|---|
| $matU | **[optional]** matrix ID of the orthogonal matrix [rowsA, rowsA] |
| $maxIter | **[optional]** maximum number of iterations for achieving convergence |

## Return Value

Success: True, or (if $enableSpecs = **True** or $maxiter > 0) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on each individual decomposition. More specific documentation on this specific decomposition can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

This function does not accept any matrix inputs of, or related to, complex types.

This decomposition normally requires a **square** input matrix **A**. Alternatively, it can take a (square) Hessenberg matrix **H** ($matH) as input, if the appropriate flag is set (i.e., $computeFromHessenberg = True). If so, it also references the associated orthogonal transformation matrix **Q**, which should then be provided directly as parsed matrix ID for $matQ. If a Hessenberg input matrix is selected *without* provding matrix **Q**, an Identity matrix will be deployed in its stead.

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Eigen's default maximum number of iterations to achieve convergence is 40 times rowsA; this particular function alternatively accepts a user-defined maximum parsed in parameter $maxiter.

## Related

*Decomp_Hessenberg(), Decomp_RealQZ(), Decomp_ComplexSchur()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "TU", True )     ; True = reset all flags first

_Eigen_Decomp_RealSchur ( $matA )   ; the decomposition call
If @error Then Exit

$matT = _Eigen_GetActiveMatrix ( "T" )    ; collect new IDs one at a time
$matU = _Eigen_GetActiveMatrix ( "U" )
```

```
_MatrixDisplay ( $matT, "RealSchur T matrix" )
_MatrixDisplay ( $matU, "RealSchur U matrix" )

_Eigen_ResetActiveMatrix_Single ( "T" )
_Eigen_ResetActiveMatrix_Single ( "U" )

_Eigen_CleanUp()
```





# _Eigen_Decomp_Tridiagonalization

Perform a Tridiagonalization decomposition

```
#include <Eigen4AutoIt.au3>
_Eigen_Decomp_Tridiagonalization ( $matA[, $enableSpecs = False[, $matC =
0[, $matD = 0[, $matP = 0[, $matQ = 0[, $matS = 0[, $matT = 0 ]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $matC | **[optional]** matrix ID of the HouseholderQR coefficients (dimensions: see Remarks) |
| $matD | **[optional]** matrix ID of the Diagonal [rowsA, 1] |
| $matP | **[optional]** matrix ID of the Packed matrix [rowsA, rowsA] |
| $matQ | **[optional]** matrix ID of the orthogonal transformation matrix [rowsA, rowsA] |
| $matS | **[optional]** matrix ID of the subDiagonal [rowsA, 1] |
| $matT | **[optional]** matrix ID of the triDiagonal matrix [rowsA, rowsA] |

## Return Value

| | |
|---|---|
| Success: | True, or (if $enableSpecs = **True**) array $decompSpecs |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of eigensolvers for a general overview and links to extensive documentation on each individual solver. More specific documentation on this specific function can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

This decomposition requires a square, real or complex input matrix **A**. Some outputs reflect the input matrix's type; others are always of type real (see table below).

The HouseholderQR coefficients matrix $matC is a Colvector of diagonal size -1.

Hessenberg and Tridiagonlization are related; both accept either a real or a complex input matrix, several optional outputs are available in both decompositions, and their type depends on the input matrix type (see table). Note that Tridiagonalization always returns matrices **D**, **T**, and **S** as type real, whereas the matrix type of **C**, **P**, and **Q** (as in the Hessenberg case) reflect the input type.

| Function | Variants | Inputs | Real Outputs | Complex Outputs |
|---|---|---|---|---|
| Hessenberg | real | real (**A**) | **C**: Householder coeffs<br>**H**: Hessenberg matrix<br>**P**: packed matrix<br>**Q**: orthogonal matrix | |
| | complex | complex (**A**) | | **C**: Householder coeffs<br>**H**: Hessenberg matrix<br>**P**: packed matrix<br>**Q**: orthogonal matrix |
| Tridiagonalization | real | real (**A**) | **C**: Householder coeffs<br>**D**: diagonal<br>**P**: packed matrix<br>**Q**: orthogonal matrix<br>**S**: subDiagonal<br>**T**: triDiagonal | |
| | complex | complex (**A**) | **D**: diagonal<br>**T**: triDiagonal<br>**S**: subDiagonal | **C**: Householder coeffs<br>**P**: packed matrix<br>**Q**: unitary matrix |

## Related

*Decomp_RealQZ*(), *Decomp_RealSchur*(), *Decomp_Hessenberg*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp( "complexfloat" )

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay_Real ( $matA, "matrix A, real" )
_MatrixDisplay_Imag ( $matA, "matrix A, imaginary" )

_Eigen_SetActiveMatrix ( "DPQS", True )   ; True = reset all flags first

_Eigen_Decomp_Tridiagonalization ( $matA )      ; the decomposition call
If @error Then Exit

$matD = _Eigen_GetActiveMatrix ( "D" )
$matP = _Eigen_GetActiveMatrix ( "P" )
$matQ = _Eigen_GetActiveMatrix ( "Q" )
$matS = _Eigen_GetActiveMatrix ( "S" )

_MatrixDisplay ( $matD, "matrix D" )
```

```
_MatrixDisplay ( $matS, "matrix S" )

_MatrixDisplay_Real ( $matP, "matrix P, real" )
_MatrixDisplay_Imag ( $matP, "matrix P, imag" )

_MatrixDisplay_Real ( $matQ, "matrix Q, real" )
_MatrixDisplay_Imag ( $matQ, "matrix Q, imag" )

_Eigen_CleanUp()
```





# _Eigen_Show_DecompSpecs

Display the list of decomposition Specs

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_DecompSpecs ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True, displayed array |
| Failure: | n/a |

## Remarks

For demonstration purposes only (listing the variables); all values are set to missing (value: -1).

## Related

*Show_MatrixSpecs(), Show_DecompSpecs_Current()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

_Eigen_Show_DecompSpecs()

_Eigen_CleanUp()
```

Show_DecompSpecs_Current

Function Reference

# _Eigen_Show_DecompSpecs_Current

Display the current contents of the decomposition Specs

```
#include <Eigen4AutoIt.au3>
_Eigen_Show_DecompSpecs_Current ()
```

## Parameters

None.

## Return Value

| | |
|---|---|
| Success: | True, displayed array |
| Failure: | n/a |

## Remarks

None.

## Related

*Show_MatrixSpecs_Current*(), *Show_DecompSpecs*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

$enablespecs = True
_Eigen_Decomp_HouseholderQR ( $matA, $enablespecs )   ; the decomposition call
If @error Then Exit

_Eigen_Show_DecompSpecs_Current()

_Eigen_CleanUp()
```



**EigenSolvers**

# EigenSolvers

Eigensolvers and related decompositions (Schur, Real QZ, Hessenberg, Tridiagonalization) are subject to special constraints on their inputs, outputs, and operational flags. Some accept only real inputs, others only complex inputs, whereas a few accept both. **Outputs can be real, complex, or mixed,** and **the matrix type of the outputs does not necessarily match the matrix type of the inputs!** Thus a real matrix supplied as input to an eigensolver *may* produce a complex result. Moreover, most of these functions can produce several additional outputs, whose type may be either always real, always complex, reflect the matrix type of the input, or depend on the actual *values* in the input matrix. Now you know why they calll them "complex numbers."

This variability in outputs poses a special challenge in terms of the work environment's matrix type. Under normal circumstances, one first selects a matrix type to work with at startup; all matrices created thereafter will be of that chosen type, and all functions called will select the associated function variant in the dlls. For eigensolvers, these conventions have to be relaxed, in that no matrix type conformity check is performed on their input matrices. So you can select a complex work environment, and then supply a real matrix as input for most eigensolvers. Obviously, since all output containers are defined prior to the actual dll function being called, the work environment still imposes restrictions on storage. In a complex environment, the consequences are trivial: if the eigensolver produces a complex output, it is stored as per usual, and if the output happens to be real, it is stored in the complex container's real part only, with the imaginary part zeroed out. However, if the matrix environment is real but the eigensolver's result is complex, the supplied real output containers cannot hold any imaginary part(s) of the result(s), triggering the dedicated status code **4**: "Complex output: unable to store imaginary part of complex output(s) in containers for reals." The real part will still be returned, but the error message will alert you to the fact that additional data may have been lost, and that changing to a complex work environment (and repeating the call there) is needed to obtain these.

Furthermore, a number of optional outputs and flags deserve mention. Eigenvalues are always returned, by default as a matrix of the same size as the input matrix, which itself is **always square**. However, if boolean flag $eigenvaluesAsVector = True (default: False), a (real or complex) column vector is returned instead. This flag also applies to **JacobiSVD**. Secondly, for iterative solvers (EigenSolver, EigenSolver_Generalized, EigenSolver_Complex, and ComplexSchur), the default maximum number of iterations to achieve convergence is 40 times the number of rows, but this parameter can be user-defined in some cases. The two SelfAdjoint eigensolvers are faster and more accurate, and can provide additional outputs sqrt(**A**) and inverse sqrt(**A**) if the input is positive-definite, but these solvers do require diagonally-symmetric inputs (it is the user's responsibility to ensure this, as it would take the wrapper functions too long to check). Finally, only function EigenSolver itself provides optional pseudo-eigenvalues (block-diagonal real matrix **D**) and pseudo-eigenvectors (real matrix **V**). See the relevant Eigen documentation for additional details.

Overview of the main properties of **EigenSolvers** and related decompositions:

| Function | Variants | Inputs | Real Outputs | Complex Outputs | MaxIter |
|---|---|---|---|---|---|
| EigenSolver | real & complex | real (**A**) | **D**: pseudo-eigenvalues<br>**V**: pseudo-eigenvectors | **S**: eigenvalues<br>**U**: eigenvectors | yes |
| EigenSolver Generalized [1] | real & complex | real (**A**,**B**) | **Q**: betas | **S**: eigenvalues<br>**P**: alphas | yes |
| EigenSolver_Generalized SelfAdjoint [2,3] | real | real (**A**,**B**) | **S**: eigenvalues<br>**U**: eigenvectors | | |

| | | | | | |
|---|---|---|---|---|---|
| | | | **R**: sqrt(A)<br>**I**: inverse sqrt(A) | | |
| SelfAdjoint EigenSolver [3] | real | real (**A**) | **S**: eigenvalues<br>**U**: eigenvectors<br>**R**: sqrt(A)<br>**I**: inverse sqrt(A) | | |
| EigenSolver Complex [4] | complex | complex (**A**) | | **S**: eigenvalues<br>**U**: eigenvectors | yes |
| Decomp Complex Schur [4] | complex | complex (**A** or **H**[,**Q**]) | | **T**: upper trian. matrix<br>**U**: unitary matrix | yes |
| Decomp Real Schur[3] | real | real (**A** or **H**[,**Q**]) | **T**: quasi-trian. matrix<br>**U**: orthogonal matrix | | yes |
| Decomp Real QZ[3] | real | real (**A**,**B**) | **Q**: 1st orthogonal matrix<br>**S**: quasi-trian. matrix<br>**T**: triangular matrix<br>**Z**: 2nd orthogonal matrix | | yes |
| Decomp Hessenberg | real | real (**A**) | **C**: Householder coeffs<br>**H**: Hessenberg matrix<br>**P**: packed matrix<br>**Q**: orthogonal matrix | | |
| | complex | complex (**A**) | | **C**: Householder coeffs<br>**H**: Hessenberg matrix<br>**P**: packed matrix<br>**Q**: orthogonal matrix | |
| Decomp Tridiagonalization | real | real (**A**) | **C**: Householder coeffs<br>**D**: diagonal<br>**P**: packed matrix<br>**Q**: orthogonal matrix<br>**S**: subDiagonal<br>**T**: triDiagonal | | |
| | complex | complex (**A**) | **D**: diagonal<br>**T**: triDiagonal<br>**S**: subDiagonal | **C**: Householder coeffs<br>**P**: packed matrix<br>**Q**: unitary matrix | |

[1]: no eigenvectors returned; [2]: three compute options: Ax_lBx, ABx_lx, BAx_lx; [3]: real inputs and outputs only; [4]:complex inputs only

Notes:
If real variants are available (see second column), dll functions are supported with suffices **_f** and **_d**.
If complex variants are available, dll functions are supported with suffices **_cf** and **_cd**.

None of the tabulated functions can act separately on either the real or imaginary part of a complex input.

The matrix type of the input(s) is **not** checked against the current work environment's type. However, it has to be complex for ComplexSchur and EigenSolver_Complex, it can be either real or complex for Hessenberg and Tridiagonalization decompositions (note the different output types depending on the input type), and has to be real for all other functions tabulated here. Submitting the wrong input type for a specific function does trigger an error message.

The matrix type of the optional outputs' *containers* is dependent on the work environment's type. If the containiner's type is real, but the actual output is complex *and its imaginary part is non-zero*, the dll status flag will be set to error **4**, indicating that information has been lost. If the container is complex and the actual output is real, the container's imaginary part is set to zero.

The last column in the table identifies whether Eigen variable ***maxIterations*** can be user-defined; parsing your own value to the dll function implies that array $decompSpecs is enabled and will be returned (instead of True/False), regardless of whether flag $enableSpecs has been explicitly set to True.

Lastly, note that three of the listed functions have special properties. Both EigenSolver and EigenSolver_Generalized produce mixed outputs from a real input, and Tridiagonalization produces mixed outputs when processing a complex input. The other functions simply reflect the input type in the output type.





# _Eigen_EigenSolver

Perform eigen analysis on a real input matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_EigenSolver ( $matA[, $enableSpecs = False[, $eigenvaluesAsVector =
False[, $matD = 0[, $matS = 0[, $matU = 0[, $matV = 0[, $maxIter = -
1 ]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $eigenvaluesAsVector | **[optional]** **True**: store **S** as Colvector; **False**: store **S** as diagonal of square matrix |
| $matD | **[optional]** matrix ID of the pseudo-eigenvalues [rowsA, rowsA] (always real) |
| $matS | **[optional]** matrix ID of the eigenvalues (dimensions: [rowsA, rowsA] or [rowsA,1]) |
| $matU | **[optional]** matrix ID of the eigenvectors [rowsA, rowsA] |
| $matV | **[optional]** matrix ID of the pseudo-eigenvectors [rowsA, rowsA] (always real) |
| $maxIter | **[optional]** maximum number of iterations for achieving convergence |

## Return Value

| | |
|---|---|
| Success: | True, or (if $enableSpecs = **True** or $maxiter > 0) array $decompSpecs |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

A separate table lists the type of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of eigensolvers for a general overview and links to extensive documentation on each individual solver. More specific documentation on this specific Eigensolver can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

All Eigensolvers require square inputs. This function expects a real matrix as input.

Outputs are mixed complex (**S**, **U**) and real (**D**, **V**). If your work environment is real, a result is complex, and its imaginary part is non-zero, then that imaginary part is lost, and the status code is set to 4. If your work environment is complex, real-only results are stored in the complex containers' real part, with the imaginary part set to zero.

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Eigen's default maximum number of iterations to achieve convergence is 40 times rowsA; this particular function alternatively accepts a user-defined maximum parsed in parameter $maxiter.

## Related

*EigenSolver_Complex*(), *EigenSolver_Generalized*(), *EigenSolver_Generalized_SelfAdjoint*(), *EigenSolver_SelfAdjoint*(), *Decomp_JacobiSVD*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "DSUV", True )   ; True = reset all flags first

_Eigen_EigenSolver ( $matA )
If @error Then Exit

$matD = _Eigen_GetActiveMatrix ( "D" )    ; collect new IDs one at a time
$matS = _Eigen_GetActiveMatrix ( "S" )
$matU = _Eigen_GetActiveMatrix ( "U" )
$matV = _Eigen_GetActiveMatrix ( "V" )

_MatrixDisplay ( $matS, "Eigenvalues" )
_MatrixDisplay ( $matU, "Eigenvectors" )
_MatrixDisplay ( $matD, "pseudo-Eigenvalues" )
_MatrixDisplay ( $matV, "psuedo-Eigenvectors" )

_Eigen_CleanUp()
```

EigenSolver_Complex

# _Eigen_EigenSolver_Complex

Perform eigen analysis on a complex input matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_EigenSolver_Complex ( $matA[, $enableSpecs = False[,
$eigenvaluesAsVector = False[, $matS = 0[, $matU = 0[, $maxIter = -1 ]]]]]
)
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $eigenvaluesAsVector | **[optional]** **True**: store **S** as Colvector; **False**: store **S** as diagonal of square matrix |
| $matS | **[optional]** matrix ID of the eigenvalues (dimensions: [rowsA, rowsA] or [rowsA,1]) |
| $matU | **[optional]** matrix ID of the eigenvectors [rowsA, rowsA] |
| $maxIter | **[optional]** maximum number of iterations for achieving convergence |

## Return Value

Success: True, or (if $enableSpecs = **True** or $maxiter > 0) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

A separate table lists the type of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of eigensolvers for a general overview and links to extensive documentation on each individual solver. More specific documentation on this specific Eigensolver can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

All Eigensolvers require square inputs. This function expects a complex matrix as input.

All outputs are always complex. If your work environment is real and a result is complex, its imaginary part is lost, and the status code is set to 4.

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Eigen's default maximum number of iterations to achieve convergence is 40 times rowsA; this particular function alternatively accepts a user-defined maximum parsed in parameter $maxiter.

## Related

*EigenSolver(), EigenSolver_Generalized(), EigenSolver_Generalized_SelfAdjoint(), EigenSolver_SelfAdjoint(), Decomp_JacobiSVD()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp( "complexfloat" )

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square, complex input
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "SU", True )     ; True = reset all flags first

_Eigen_EigenSolver_Complex ( $matA )
If @error Then Exit

$matS = _Eigen_GetActiveMatrix ( "S" )
$matU = _Eigen_GetActiveMatrix ( "U" )

_MatrixDisplay_Real ( $matS, "Eigenvalues, real" )
_MatrixDisplay_Imag ( $matS, "Eigenvalues, imaginary" )

_MatrixDisplay_Real ( $matU, "Eigenvectors, real" )
_MatrixDisplay_Imag ( $matU, "Eigenvectors, imaginary" )

_Eigen_CleanUp()
```



EigenSolver_Generalized

# _Eigen_EigenSolver_Generalized

Perform generalized eigen analysis on two input matrices

```
#include <Eigen4AutoIt.au3>
_Eigen_EigenSolver_Generalized ( $matA, $matB[, $enableSpecs = False[,
$eigenvaluesAsVector = False[, $matD = 0[, $matS = 0[, $matU = 0[, $matV =
0[, $maxIter = -1 ]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix (dimensions: [rowsA, rowsA], square) |
| $matB | matrix ID of the second input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $eigenvaluesAsVect | **[optional]** **True**: store **S** as Colvector; **False**: store **S** as diagonal of square matrix |

| | |
|---|---|
| or | |
| $matP | **[optional]** matrix ID of the alpha values [rowsA, 1] |
| $matQ | **[optional]** matrix ID of the beta values [rowsA, 1] |
| $matS | **[optional]** matrix ID of the eigenvalues (dimensions: [rowsA, rowsA] or [rowsA,1]) |
| $maxIter | **[optional]** maximum number of iterations for achieving convergence |

## Return Value

| | |
|---|---|
| Success: | True, or (if $enableSpecs = **True** or $maxiter > 0) array $decompSpecs |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

A separate table lists the type of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of eigensolvers for a general overview and links to extensive documentation on each individual solver. More specific documentation on this specific Eigensolver can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

All Eigensolvers require square inputs. This function expects two equal-sized **real** inputs.

This function does not accept any complex matrix inputs.

Outputs are mixed complex (**P**, **S**) and real (**Q**). If your work environment is complex, real-only results are stored in the complex containers' real part.

Eigen limitation: this function does not return eigenvectors (yet).

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Eigen's default maximum number of iterations to achieve convergence is 40 times rowsA; this particular function alternatively accepts a user-defined maximum parsed in parameter $maxiter.

## Related

*EigenSolver*(), *EigenSolver_Complex*(), *EigenSolver_Generalized_SelfAdjoint*(), *EigenSolver_SelfAdjoint*(), *Decomp_JacobiSVD*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp( "complexfloat" )    ; complex work environment

$matA = _Eigen_CreateMatrix_Random ( 6, 6, "realfloat" )    ; real(!) square
input A
_MatrixDisplay ( $matA, "real matrix A" )
```

```
$matB = _Eigen_CreateMatrix_Random ( 6, 6, "realfloat" )    ; real(!) square
input B
_MatrixDisplay ( $matB, "real matrix B" )

_Eigen_SetActiveMatrix ( "PQS", True )    ; True = reset all flags first

_Eigen_EigenSolver_Generalized ( $matA, $matB )
If @error Then Exit

$matS = _Eigen_GetActiveMatrix ( "S" )    ; eigenvalues (eigenvectors are not
returned)
$matP = _Eigen_GetActiveMatrix ( "P" )    ; complex numerators (alphas)
$matQ = _Eigen_GetActiveMatrix ( "Q" )    ; real denominators (betas)

_MatrixDisplay_Real ( $matS, "Eigenvalues, real" )
_MatrixDisplay_Imag ( $matS, "Eigenvalues, imag" )

_MatrixDisplay_Real ( $matP, "alphas, real" )
_MatrixDisplay_Imag ( $matP, "alphas, imag" )

_MatrixDisplay ( $matQ, "betas" )

_Eigen_CleanUp()
```





# _Eigen_EigenSolver_Generalized_SelfAdjoint

Perform generalized eigen analysis on a symmetric matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_EigenSolver_Generalized_SelfAdjoint ( $matA, $matB[, $enableSpecs =
False[, $eigenvaluesAsVector = False[, $ABx_lambdaX = False[, $BAx_lambdaX
= False[, $matI = 0[, $matR = 0[, $matS = 0[, $matU = 0 ]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix (dimensions: [rowsA, rowsA], square, **symmetric**) |
| $matB | matrix ID of the second input matrix (dimensions: [rowsA, rowsA], square) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $eigenvaluesAsVector | **[optional]** **True**: store **S** as Colvector; **False**: store **S** as diagonal of square matrix |
| $ABx_lambdaX | **[optional]** **True**: solve **A.B.x** = λ.**x** instead of **A.x** = λ.**B.x**; **False**: don't |
| $BAx_lambdaX | **[optional]** **True**: solve **B.A.x** = λ.**x** instead of **A.x** = λ.**B.x**; **False**: don't |
| $matI | **[optional]** matrix ID of the inverse square root of **A** [rowsA, rowsA] (always real) |

| | |
|---|---|
| $matR | **[optional]** matrix ID of the square root of **A** [rowsA, rowsA] (always real) |
| $matS | **[optional]** matrix ID of the eigenvalues (dimensions: [rowsA, rowsA] or [rowsA,1]) |
| $matU | **[optional]** matrix ID of the eigenvectors [rowsA, rowsA] |

## Return Value

Success: True, or (if $enableSpecs = **True**) array $decompSpecs

Failure: False, and sets the @error flag to non-zero.

## Remarks

A separate table lists the type of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of eigensolvers for a general overview and links to extensive documentation on each individual solver. More specific documentation on this specific Eigensolver can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

All Eigensolvers require square inputs. This function expects two equal-sized real inputs; the first one should be symmetric.

**BEWARE**: If optional output matrix **I** and/or matrix **R** is activated ((inverse) sqrt(**A**)), then input matrix **A** has to be **positive-definite** as well. It is the user's responsibility to ensure this.

All outputs are real. If your work environment is complex, real-only results are stored in the complex containers' real part.

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Eigen's default maximum number of iterations to achieve convergence is 40 times rowsA; this particular function does not accept a user-defined maximum.

## Related

*EigenSolver*(), *EigenSolver_Complex*(), *EigenSolver_Generalized*(), *EigenSolver_SelfAdjoint*(), *Decomp_JacobiSVD*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_Eigen_Multiply_AAt_InPlace ( $matA )     ; make A symmetric, positive-
definite
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix_Random ( 6, 6, "realfloat" )    ; real(!) square
input B
_MatrixDisplay ( $matB, "real matrix B" )

_Eigen_SetActiveMatrix ( "SURI", True )   ; True = reset all flags first
```

```
_Eigen_EigenSolver ( $matA )
If @error Then Exit

$matS = _Eigen_GetActiveMatrix ( "S" )
$matU = _Eigen_GetActiveMatrix ( "U" )
$matR = _Eigen_GetActiveMatrix ( "R" )
$matI = _Eigen_GetActiveMatrix ( "I" )

_MatrixDisplay ( $matS, "Eigenvalues" )
_MatrixDisplay ( $matU, "Eigenvectors" )
_MatrixDisplay ( $matR, "sqrt(A)" )
_MatrixDisplay ( $matI, "inverse srqt(A)" )

_Eigen_CleanUp()
```





# _Eigen_EigenSolver_SelfAdjoint

Perform eigen analysis on a symmetric matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_EigenSolver_SelfAdjoint ( $matA[, $enableSpecs = False[,
$eigenvaluesAsVector = False[, $matI = 0[, $matR = 0[, $matS = 0[, $matU =
0 ]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, rowsA], square, **symmetric**) |
| $enableSpecs | **[optional]** **True**: return array $decompSpecs; **False**: don't |
| $eigenvaluesAsVector | **[optional]** **True**: store **S** as Colvector; **False**: store **S** as diagonal of square matrix |
| $matI | **[optional]** matrix ID of the inverse square root of **A** [rowsA, rowsA] (always real) |
| $matR | **[optional]** matrix ID of the square root of **A** [rowsA, rowsA] (always real) |
| $matS | **[optional]** matrix ID of the eigenvalues (dimensions: [rowsA, rowsA] or [rowsA,1]) |
| $matU | **[optional]** matrix ID of the eigenvectors [rowsA, rowsA] |

## Return Value

| | |
|---|---|
| Success: | True, or (if $enableSpecs = **True**) array $decompSpecs |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

A separate table lists the type of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of eigensolvers for a general overview and links to extensive documentation on each individual solver. More specific documentation on this specific Eigensolver can be found here.

This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

All Eigensolvers require square inputs.  This function expects a real, **symmetric** matrix as input.

**BEWARE**: If optional output matrix **I** and/or matrix **R** is activated ((inverse) sqrt(**A**)), then input matrix **A** has to be **positive-definite** as well. It is the user's responsibility to ensure this.

All outputs are real. If your work environment is complex, real-only results are stored in the complex containers' real part.

This decomposition relies on an iterative algorithm. Its convergence speed depends on how well the eigenvalues are separated. Eigen's default maximum number of iterations to achieve convergence is 40 times rowsA; this particular function does not accept a user-defined maximum.

## Related

*EigenSolver*(), *EigenSolver_Complex*(), *EigenSolver_Generalized*(), *EigenSolver_Generalized_SelfAdjoint*(), *Decomp_JacobiSVD*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_Eigen_Multiply_AAt_InPlace ( $matA )     ; make A symmetric, positive-
definite
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "SURI", True )   ; True = reset all flags first

_Eigen_EigenSolver ( $matA )
If @error Then Exit

$matS = _Eigen_GetActiveMatrix ( "S" )
$matU = _Eigen_GetActiveMatrix ( "U" )
$matR = _Eigen_GetActiveMatrix ( "R" )
$matI = _Eigen_GetActiveMatrix ( "I" )

_MatrixDisplay ( $matS, "Eigenvalues" )
_MatrixDisplay ( $matU, "Eigenvectors" )
_MatrixDisplay ( $matR, "sqrt(A)" )
_MatrixDisplay ( $matI, "inverse srqt(A)" )

_Eigen_CleanUp()
```

## Statistics

# Statistics

This module is not provided by **Eigen**, but implemented in **Eigen4AutoIt** by the author, because let's face it, who can live without standard deviation (StDev), centering (mean-subtraction), normalisation (division by StDev or sum of absolute values), correlation, and covariance? Likewise, it would be nice to be able to fit a general linear *model* (which is not necessarily the same as fitting a straight line!), compute its misfit, or obtain a matrix's principal components without a fuss. To explore the latter two additions, please check out EigenTutorials Regression and PCA. Statistics functions are not supported for complex inputs.

Unbiased (the default) and biased versions of the following functions are defined as follows:

| **Function** | **unbiased** | **biased** |
|---|---|---|
| Standard Deviation (StDev) | $s = \sqrt{\frac{1}{(N-1)}\sum_{i=1}^{N}(x_i - \bar{x})^2}$ | $s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$ |
| Skewne | $\gamma = \sqrt{\frac{N(N-1)}{N-2}}\left(\frac{\sum_{i=1}^{N}[(x-\bar{x})^3]/N}{\left[\sum_{i=1}^{N}[(x-\bar{x})^2]/N\right]^{3/2}}\right)$ | $\gamma = \left(\frac{\sum_{i=1}^{N}[(x-\bar{x})^3]/N}{\left[\sum_{i=1}^{N}[(x-\bar{x})^2]/N\right]^{3/2}}\right)$ |

| | | |
|---|---|---|
| Skewness | $k = \frac{(N-1)}{(N-2)(N-3)}\left[(N+1)\left\{\left(\frac{\sum_{i=1}^{N}[(x-\bar{x})^4]/N}{\left[\sum_{i=1}^{N}[(x-\bar{x})^2]/N\right]^2}\right)-3\right\}\right]+6$ | $k = \left(\frac{\sum_{i=1}^{N}[(x-\bar{x})^4]/N}{\left[\sum_{i=1}^{N}[(x-\bar{x})^2]/N\right]^2}\right)-3$ |
| Kurtosis | | |
| Covariance | $Q_{jk} = \frac{1}{(N-1)}\sum_{i=1}^{N}(x_{ij}-\bar{x_j})(x_{ik}-\bar{x_k})$ | $Q_{jk} = \sum_{i=1}^{N}(x_{ij}-\bar{x_j})(x_{ik}-\bar{x_k})$ |
| Correlation | $r = \frac{\sum_{i=1}^{N}(A_i-\bar{A})(B_i-\bar{B})}{(N-1)s_A s_B}$ | $r = \frac{\sum_{i=1}^{N}(A_i-\bar{A})(B_i-\bar{B})}{N s_A s_B}$ |

In all formulae tabulated here, *N* denotes the total number of observations, *i, j, k* are observation indices, and overbar denotes the mean.

The unbiased c.q. biased versions of normalisation (see _Eigen_Normalise()) and correlation use, respectively, the unbiased or biased version of the standard deviation ***s*** as defined here. Normalisation can also be applied Colwise or Rowwise, and may optionally be stored in-place.

The kurtosis defined here is the so-called excess kurtosis, which expresses peakedness with respect to a Gaussian distribution through the -3 term.

The standard form of covariance matrix **Q** operates Colwise (in which rows are observations, columns are variables); the non-standard, Rowwise form (in which rows are variables, columns are observations) is also provided; **Q** is a k-by-k matrix in which each entry $Q_{jk}$ is an estimate of the covariance between the *j*-th and the *k*-th variable of the population. Note finally that the biased form of the skewness has no population size prefactor, and the biased form of the covariance has no divisor.



## Center

Function Reference

# _Eigen_Center

Cellwise-subtract the global matrix mean, so all values center around zero

```
#include <Eigen4AutoIt.au3>
_Eigen_Center ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If no existing results matrix **R** of the correct dimensions is supplied (in this case, the same dimensions as input matrix **A**), a new results matrix will be created.

## Related

*Center_Colwise*(), *Center_Colwise_InPlace*(), *Center_InPlace*(), *Center_Rowwise*(), *Center_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Center ( $matA )
_MatrixDisplay ( $matR, "A centered" )

_Eigen_CleanUp()
```

# _Eigen_Center_Colwise

Cellwise-subtract the mean per column, so all values center around zero per column

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_Colwise ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If no existing results matrix **R** of the correct dimensions is supplied (in this case, the same dimensions as input matrix **A**), a new results matrix will be created.

## Related

*Center*(), *Center_Colwise_InPlace*(), *Center_InPlace*(), *Center_Rowwise*(), *Center_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )      ; square
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Center_Colwise ( $matA )
_MatrixDisplay ( $matR, "A centered Colwise" )

_Eigen_CleanUp()
```



### Center_Colwise_InPlace

Function Reference

# _Eigen_Center_Colwise_InPlace

Cellwise-subtract the mean per column, so all values center around zero per column; store in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_Colwise_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Center*(), *Center_Colwise*(), *Center_InPlace*(), *Center_Rowwise*(), *Center_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Center_Colwise_InPlace ( $matA )
_MatrixDisplay ( $matA, "A centered Colwise" )

_Eigen_CleanUp()
```

# _Eigen_Center_InPlace

Cellwise-subtract the global matrix mean, so all values center around zero; store in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Center*(), *Center_Colwise*(), *Center_Colwise_InPlace*(), *Center_Rowwise*(), *Center_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Center_InPlace ( $matA )
_MatrixDisplay ( $matA, "A centered" )

_Eigen_CleanUp()
```

# _Eigen_Center_Rowwise

Cellwise-subtract the mean per row, so all values center around zero per row

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_Rowwise ( $matA[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If no existing results matrix **R** of the correct dimensions is supplied (in this case, the same dimensions as input matrix **A**), a new results matrix will be created.

## Related

*Center*(), *Center_Colwise*(), *Center_Colwise_InPlace*(), *Center_InPlace*(), *Center_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Center_Rowwise ( $matA )
_MatrixDisplay ( $matR, "A centered Rowwise" )

_Eigen_CleanUp()
```

# _Eigen_Center_Rowwise_InPlace

Cellwise-subtract the mean per row, so all values center around zero per row; store in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_Rowwise_InPlace ( $matA )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

The results are stored in the input matrix, replacing the original data.

## Related

*Center*(), *Center_Colwise*(), *Center_Colwise_InPlace*(), *Center_InPlace*(), *Center_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Center_Rowwise_InPlace ( $matA )
_MatrixDisplay ( $matA, "A centered Rowwise" )

_Eigen_CleanUp()
```

# _Eigen_Correlation_AB

Compute the correlation coefficient for two same-sized matrices

```
#include <Eigen4AutoIt.au3>
_Eigen_Correlation_AB ( $matA, $matB[, $unbiased = True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the first input matrix |
| $matB | matrix ID of the second input matrix |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | value (correlation coefficient) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The correlation provides a measure of how much two equal-sized data sets (here: matrices) vary linearly together, normalised by the standard deviation. The returned coefficient ranges in value from **+1** (perfect positive correlation) through **0** (no correlation whatsoever) to **-1** (perfect negative correlation).

Minimum size for the input matrix: 3 cells.

## Related

*Correlation_Col_Col()*, *Correlation_Row_Row()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 5 )
_MatrixDisplay ( $matA, "matrix A" )

$matB = _Eigen_CreateMatrix_Random ( 4, 5 )
_MatrixDisplay ( $matB, "matrix B" )

MSgBox ( 0, "matrices A and B", "Correlation coefficient = " & _
      _Eigen_Correlation_AB ( $matA, $matB ) )

_Eigen_CleanUp()
```

# _Eigen_Correlation_ColCol

Compute the correlation coefficient for two columns in a matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Correlation_ColCol ( $matA, $colindex1, $colindex2[, $unbiased =
True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $colindex1 | column ID (base-0) of the first column in matrix A to act upon |
| $colindex2 | column ID (base-0) of the second column in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (correlation coefficient)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The correlation provides a measure of how much two equal-sized data sets (here: columns) vary linearly together, normalised by the standard deviation. This coefficient ranges in value from **+1** (perfect positive correlation) through **0** (no correlation whatsoever) to **-1** (perfect negative correlation).

Minimum size for the input matrix: 3 rows.

## Related

*Correlation_AB(), Correlation_RowRow()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 8, 4 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
$colindex1 = 0
$colindex2 = 2

MSgBox ( 0, "matrix A", "Correlation coefficient for columns " & _
      $colindex1 & " and " & $colindex2 & " = " & _
      _Eigen_Correlation_ColCol ( $matA, $colindex1, $colindex2 ) )

_Eigen_CleanUp()
```





# _Eigen_Correlation_RowRow

Compute the correlation coefficient for two rows in a matrix

```
#include <Eigen4AutoIt.au3>
_Eigen_Correlation_RowRow ( $matA, $rowindex1, $rowindex2[, $unbiased =
True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $rowindex1 | row ID (base-0) of the first row in matrix A to act upon |
| $rowindex2 | row ID (base-0) of the second row in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (correlation coefficient)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The correlation provides a measure of how much two equal-sized data sets (here: rows) vary linearly together, normalised by the standard deviation. This coefficient ranges in value from **+1** (perfect positive correlation) through **0** (no correlation whatsoever) to **-1** (perfect negative correlation).

Minimum size for the input matrix: 3 columns.

## Related

*Correlation_AB(), Correlation_ColCol()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 8 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex1 = 0
$rowindex2 = 2
MSgBox ( 0, "matrix A", "Correlation coefficient for rows " & _
      $rowindex1 & " and " & $rowindex2 & " = " & _
      _Eigen_Correlation_RowRow ( $matA, $rowindex1, $rowindex2 ) )

_Eigen_CleanUp()
```





# _Eigen_Covariance_Colwise

Compute the covariance matrix (standard form)

```
#include <Eigen4AutoIt.au3>
_Eigen_Covariance_Colwise ( $matA[, $matR = 0[, $unbiased = True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the covariance matrix |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: $matR (covariance matrix)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The covariance provides a measure of how much pairs of variables (here: columns) vary linearly together. Here, each column is interpreted as a variable, the rows as joint measurements. The strength and direction of the relationship between each pair of columns is then stored in the (square) covariance matrix, with the diagonal containing the variance (squared standard deviation) of the associated column, and the off-

diagonal cells the covariance for the two columns whose index matches the covariance matrix cell coordinates.

**Example**: an input matrix has 3 columns (variables) and 10 rows (observations); the square covariance matrix is then 3 x 3 cells. Diagonal cell [0,0] contains the variance for column 0, off-diagonal cell [0,1] the covariance between columns 0 and 1, off-diagonal cell [0,2] the covariance between columns 0 and 2, diagonal cell [1,1] contains the variance for column 1, and so forth. Values are mirrored on both sides of the diagonal.

Minimum size for the input matrix: 2 rows and 2 columns.

If no existing results matrix **R** of the correct dimensions is supplied (in this case, a square matrix whose dimensions match the number of columns of input matrix **A**), a new results matrix will be created.

## Related

*Covariance_Rowwise()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 8, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$matC = _Eigen_Covariance_Colwise ( $matA )
_MatrixDisplay ( $matC, "Covariance matrix" )

_Eigen_CleanUp()
```





# _Eigen_Covariance_Rowwise

Compute the covariance matrix (non-standard, transposed form)

```
#include <Eigen4AutoIt.au3>
_Eigen_Covariance_Rowwise ( $matA[, $matR = 0[, $unbiased = True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the covariance matrix |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | $matR (covariance matrix) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The covariance provides a measure of how much variable pairs (here: rows) vary linearly together. Here, each row is interpreted as a variable, the columns as joint measurements; this is the non-standard form (the standard version has variables in columns, observations in rows). The strength and direction of the relationship between each pair of rows is then stored in the (square) covariance matrix, with the diagonal containing the variance (squared standard deviation) of the associated row, and the off-diagonal cells the covariance for the two rows whose index matches the covariance matrix cell coordinates.

**Example**: an input matrix has 3 rows (variables) and 10 columns (observations); the square covariance matrix is then 3 x 3 cells. Diagonal cell [0,0] contains the variance for row 0, off-diagonal cell [0,1] the covariance between rows 0 and 1, off-diagonal cell [0,2] the covariance between rows 0 and 2, and so forth. Values are mirrored on both sides of the diagonal.

Minimum size for the input matrix: 2 rows and 2 columns.

If no existing destination matrix of the correct dimensions is supplied (in this case, a square matrix whose dimensions match the number of rows of input matrix **A**), a new results matrix will be created.

## Related

*Covariance_Colwise()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 3, 6 )
_MatrixDisplay ( $matA, "matrix A" )

$matC = _Eigen_Covariance_Rowwise ( $matA )
_MatrixDisplay ( $matC, "Covariance matrix (Rowwise)" )

_Eigen_CleanUp()
```





# _Eigen_Kurtosis

Return the global matrix kurtosis (peakedness)

```
#include <Eigen4AutoIt.au3>
_Eigen_Kurtosis ( $matA[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (kurtosis for all matrix cells combined)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Kurtosis (formally: excess kurtosis) provides a measure of the data's peakedness with respect to a Gaussian distribution.

Minimum size for the input matrix: 4 cells.

## Related

*Kurtosis_Col*(), *Kurtosis_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

MSgBox ( 0, "matrix A", "Kurtosis = " & _Eigen_Kurtosis ( $matA ) )

_Eigen_CleanUp()
```





# _Eigen_Kurtosis_Col

Return the kurtosis (peakedness) for a specified matrix column

```
#include <Eigen4AutoIt.au3>
_Eigen_Kurtosis_Col ( $matA, $colindex[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $colindex | column ID (base-0) of the column in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (kurtosis for the specified column)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Kurtosis (formally: excess kurtosis) provides a measure of the data's peakedness with respect to a Gaussian distribution.

Minimum size for the input matrix: 4 rows

## Related

*Kurtosis*(), *Kurtosis_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$colindex = 2

MSgBox ( 0, "matrix A", "Kurtosis for col " & $colindex & _
            " = " & _Eigen_Kurtosis_Col ( $matA, $colindex ) )

_Eigen_CleanUp()
```

# _Eigen_Kurtosis_Row

Return the kurtosis (peakedness) for a specified matrix row

```
#include <Eigen4AutoIt.au3>
_Eigen_Kurtosis_Col ( $matA, $rowindex[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (kurtosis for the specified row)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Kurtosis (formally: excess kurtosis) provides a measure of the data's peakedness with respect to a Gaussian distribution.

Minimum size for the input matrix: 4 columns.

## Related

*Kurtosis*(), *Kurtosis_Col*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2

MSgBox ( 0, "matrix A", "Kurtosis for row " & $rowindex & _
            " = " & _Eigen_Kurtosis_Row ( $matA, $rowindex ) )

_Eigen_CleanUp()
```

## Normalise

Function Reference

# _Eigen_Normalise

Cellwise-divide by the global matrix standard deviation or the sum of absolute values

```
#include <Eigen4AutoIt.au3>
_Eigen_Normalise ( $matA[, $matR = 0[, $useStDev = True[, $unbiased = True
]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |
| $useStDev | **[optional]** **True**: divide by standard deviation; **False**: divide by the sum total of absolute values |
| $unbiased | **[optional]** **True**: (StDev only) subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If no existing results matrix **R** of the correct dimensions is supplied (in this case, the same dimensions as input matrix **A**), a new results matrix will be created.

Minimum size for the input matrix: 2 cells.

## Related

*Normalise_Colwise*(), *Normalise_Colwise_InPlace*(), *Normalise_InPlace*(), *Normalise_Rowwise*(), *Normalise_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
$matR = _Eigen_Normalise ( $matA )
_MatrixDisplay ( $matR, "A normalised" )

_Eigen_CleanUp()
```





# _Eigen_Normalise_Colwise

Cellwise-divide by the standard deviation (or the sum of absolute values) per column, so all values are normalised per column

```
#include <Eigen4AutoIt.au3>
_Eigen_Normalise_Colwise ( $matA[, $matR = 0[, $useStDev = True[,
$unbiased = True ]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |
| $useStDev | **[optional]** **True**: divide by standard deviation; **False**: divide by the sum total of absolute values |
| $unbiased | **[optional]** **True**: (StDev only) subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | $matR |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

If no existing results matrix **R** of the correct dimensions is supplied (in this case, the same dimensions as input matrix **A**), a new results matrix will be created.

Minimum size for the input matrix: 2 rows.

## Related

*Normalise*(), *Normalise_Colwise_InPlace*(), *Normalise_InPlace*(), *Normalise_Rowwise*(), *Normalise_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Normalise_Colwise ( $matA )
_MatrixDisplay ( $matR, "A normalised Colwise" )

_Eigen_CleanUp()
```





# _Eigen_Normalise_Colwise_InPlace

Cellwise-divide by the standard deviation (or the sum of absolute values) per column, so all values are normalised per column; store in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_Colwise_InPlace ( $matA[, $useStDev = True[, $unbiased =
True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $useStDev | **[optional]** **True**: divide by standard deviation; **False**: divide by the sum total of absolute values |
| $unbiased | **[optional]** **True**: (StDev only) subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Minimum size for the input matrix: 2 cells.

The results are stored in the input matrix, replacing the original data.

## Related

*Normalise*(), *Normalise_Colwise*(), *Normalise_InPlace*(), *Normalise_Rowwise*(), *Normalise_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Normalise_Colwise_InPlace ( $matA )
_MatrixDisplay ( $matA, "A normalised Colwise" )

_Eigen_CleanUp()
```



### Normalise_InPlace

Function Reference

# _Eigen_Normalise_InPlace

Cellwise-divide by the global matrix standard deviation or the sum of absolute values; store in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_InPlace ( $matA[, $useStDev = True[, $unbiased = True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $useStDev | **[optional]** **True**: divide by standard deviation; **False**: divide by the sum total of absolute values |
| $unbiased | **[optional]** **True**: (StDev only) subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | $matA |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

Minimum size for the input matrix: 2 cells.

The results are stored in the input matrix, replacing the original data.

## Related

*Normalise*(), *Normalise_Colwise*(), *Normalise_Colwise_InPlace*(), *Normalise_Rowwise*(), *Normalise_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_ColMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Normalise_InPlace ( $matA )
_MatrixDisplay ( $matA, "A normalised" )

_Eigen_CleanUp()
```



### Normalise_Rowwise

Function Reference

# _Eigen_Normalise_Rowwise

Cellwise-divide by the standard deviation (or the sum of absolute values) per row, so all values are normalised per row

```
#include <Eigen4AutoIt.au3>
_Eigen_Normalise_Rowwise ( $matA[, $matR = 0[, $useStDev = True[,
$unbiased = True ]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results matrix |
| $useStDev | **[optional]** **True**: divide by standard deviation; **False**: divide by the sum total of absolute values |

| | |
|---|---|
| $unbiased | **[optional]** **True**: (StDev only) subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: $matR

Failure: False, and sets the @error flag to non-zero.

## Remarks

If no existing results matrix **R** of the correct dimensions is supplied (in this case, the same dimensions as input matrix **A**), a new results matrix will be created.

Minimum size for the input matrix: 2 columns.

## Related

*Normalise*(), *Normalise_Colwise*(), *Normalise_Colwise_InPlace*(), *Normalise_InPlace*(), *Normalise_Rowwise_InPlace*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_Normalise_Rowwise ( $matA )
_MatrixDisplay ( $matR, "A normalised Rowwise" )

_Eigen_CleanUp()
```





# _Eigen_Normalise_Rowwise_InPlace

Cellwise-divide by the standard deviation (or the sum of absolute values) per row, so all values are normalised per row; store in-place

```
#include <Eigen4AutoIt.au3>
_Eigen_Center_Rowwise_InPlace ( $matA[, $useStDev = True[, $unbiased =
True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $useStDev | **[optional]** **True**: divide by standard deviation; **False**: divide by the sum total of absolute values |
| $unbiased | **[optional]** **True**: (StDev only) subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: $matA

Failure: False, and sets the @error flag to non-zero.

## Remarks

Minimum size for the input matrix: 2 cells.

The results are stored in the input matrix, replacing the original data.

The unbiased version of the standard deviation is used.

## Related

*Normalise*(), *Normalise_Colwise*(), *Normalise_Colwise_InPlace*(), *Normalise_InPlace*(), *Normalise_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix ( 4, 4 )
_Eigen_SetLinSpaced_RowMajor ( $matA, 1, 16 )
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_Normalise_Rowwise_InPlace ( $matA )
_MatrixDisplay ( $matA, "A normalised Rowwise" )

_Eigen_CleanUp()
```



### Skewness

Function Reference

# _Eigen_Skewness

Return the global matrix skewness (asymmetry)

```
#include <Eigen4AutoIt.au3>
_Eigen_Skewness ( $matA[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (skewness for all matrix cells combined)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Skewness provides a measure of the data's lack of symmetry, as a departure from zero.

Minimum size for the input matrix: 3 cells.

## Related

*Skewness_Col*(), *Skewness_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

MSgBox ( 0, "matrix A", "Skewness = " & _Eigen_Skewness ( $matA ) )

_Eigen_CleanUp()
```





# _Eigen_Skewness_Col

Return the skewness (asymmetry) for a specified matrix column

```
#include <Eigen4AutoIt.au3>
_Eigen_Skewness_Col ( $matA, $colindex[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $colindex | column ID (base-0) of the column in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (skewness for the specified column)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Skewness provides a measure of the data's lack of symmetry, as a departure from zero.

Minimum size for the input matrix: 3 rows.

## Related

*Skewness*(), *Skewness_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$colindex = 2

MSgBox ( 0, "matrix A", "Skewness for col " & $colindex & _
            " = " & _Eigen_Skewness_Col ( $matA, $colindex ) )

_Eigen_CleanUp()
```



### Skewness_Row

Function Reference

# _Eigen_Skewness_Row

Return the skewness (asymmetry) for a specified matrix row

```
#include <Eigen4AutoIt.au3>
_Eigen_Skewness_Row ( $matA, $rowindex[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (skewness for the specified row)

Failure: False, and sets the @error flag to non-zero.

## Remarks

Skewness provides a measure of the data's lack of symmetry, as a departure from zero.

Minimum size for the input matrix: 3 columns.

## Related

*Skewness(), Skewness_Col()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2

MSgBox ( 0, "matrix A", "Skewness for row " & $rowindex & _
            " = " & _Eigen_Skewness_Row ( $matA, $rowindex ) )

_Eigen_CleanUp()
```



StDev

# _Eigen_StDev

Return the global matrix standard deviation

```
#include <Eigen4AutoIt.au3>
_Eigen_StDev ( $matA[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (standard deviation of all matrix cells combined)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The standard deviation provides a measure of the data's spread around the mean.

Minimum size for the input matrix: 2 cells.

## Related

*StDev_Col*(), *StDev_Colwise*(), *StDev_Row*(), *StDev_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

MSgBox ( 0, "matrix A", "Standard Deviation = " & _Eigen_StDev ( $matA ) )

_Eigen_CleanUp()
```

# _Eigen_StDev_Col

Return the standard deviation for a specified matrix column

```
#include <Eigen4AutoIt.au3>
_Eigen_StDev_Col ( $matA, $colindex[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $colindex | column ID (base-0) of the column in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

Success: value (standard deviation of the specified column)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The standard deviation provides a measure of the data's spread around the mean.

Minimum size for the input matrix: 2 rows.

## Related

*StDev()*, *StDev_Colwise()*, *StDev_Row()*, *StDev_Rowwise()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$colindex = 2

MSgBox ( 0, "matrix A", "Standard Deviation for col " & $colindex & _
            " = " & _Eigen_StDev_Col ( $matA, $colindex ) )

_Eigen_CleanUp()
```

StDev_Colwise

Function Reference

# _Eigen_StDev_Colwise

Return, in a Rowvector, the standard deviation for each column

```
#include <Eigen4AutoIt.au3>
_Eigen_StDev_Colwise ( $matA[, $matR[, $unbiased = True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results **Rowvector** |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | $matR (Rowvector of standard deviations per column) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The standard deviation provides a measure of the data's spread around the mean.

Minimum size for the input matrix: 2 rows.

If no existing results matrix **R** of the correct dimensions is supplied (in this case, a Rowvector with as many columns as input matrix **A**), a new results matrix will be created.

## Related

*StDev*(), *StDev_Col*(), *StDev_Row*(), *StDev_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$matR = _Eigen_StDev_Colwise ( $matA )
_MatrixDisplay ( $matR, "A StDev per col" )

_Eigen_CleanUp()
```

# _Eigen_StDev_Row

Return the standard deviation for a specified matrix row

```
#include <Eigen4AutoIt.au3>
_Eigen_StDev_Row ( $matA, $rowindex[, $unbiased = True ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $rowindex | row ID (base-0) of the row in matrix A to act upon |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | value (standard deviation of the specified row) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The standard deviation provides a measure of the data's spread around the mean.

Minimum size for the input matrix: 2 columns.

## Related

*StDev*(), *StDev_Col*(), *StDev_Colwise*(), *StDev_Rowwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )

$rowindex = 2

MSgBox ( 0, "matrix A", "Standard Deviation for row " & $rowindex & _
            " = " & _Eigen_StDev_Row ( $matA, $rowindex ) )
```

```
_Eigen_CleanUp()
```



StDev_Rowwise

Function Reference

# _Eigen_StDev_Rowwise

Return, in a Colvector, the standard deviation for each row

```
#include <Eigen4AutoIt.au3>
_Eigen_StDev_Rowwise ( $matA[, $matR[, $unbiased = True ]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix |
| $matR | **[optional]** matrix ID of the results **Colvector** |
| $unbiased | **[optional]** **True**: subtract one degree of freedom for the mean; **False**: don't |

## Return Value

| | |
|---|---|
| Success: | $matR (Colvector of standard deviations per row) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The standard deviation provides a measure of the data spread around the mean.

Minimum size for the input matrix: 2 rows.

If no existing results matrix **R** of the correct dimensions is supplied (in this case, a Colvector with as many rows as input matrix **A**), a new results matrix will be created.

## Related

*StDev*(), *StDev_Col*(), *StDev_Colwise*(), *StDev_Row*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 4, 4 )
_MatrixDisplay ( $matA, "matrix A" )
```

```
$matR = _Eigen_StDev_Rowwise ( $matA )
_MatrixDisplay ( $matR, "StDev per row" )

_Eigen_CleanUp()
```



Advanced

# Advanced Statistics

- LeastSquares (Gaussian linear solver)
- Misfit (sum of Absolute Residuals)
- Misfit_Col
- Misfit_Colwise
- Partial Least Squares
- PCA (Principal Components Analysis)
- PCA_ReDim (lossy compression)
- RelativeError
- RelativeError_Col
- RelativeError_colwise
- SSR (sum of Squared Residuals)
- SSR_Col
- SSR_Colwise
- Rsquared (percentage of variability accounted for)
- Rsquared_Col
- Rsquared_Colwise

**Limitation:** these functions do not accept any matrix inputs of, or related to, complex types.

In addition to the fairly slow linear solvers that are part of the main decompositions, a much faster, traditional Gaussian least-squares regression procedure is also provided. It minimises the Euclidean distance between model prediction and observations; if we write the error vector **e = A.X - B**, then we can minimise **e** to obtain the normal equations: $\mathbf{(A^T.A).X = (A^T.B)}$, which can be rearranged to solve for $\mathbf{X = (A^T.A)^{-1} . (A^T.B)}$. Unlike the decompositional linear solvers, the LeastSquares function also supports the optional inclusion of a weighting matrix **W** and/or a damping constant λ (lambda). See _Eigen_LeastSquares() for additional details.

The various ways of fitting a linear model to data are furthermore explored in Tutorial Regression.

Given a model fit **X**, obtained either through least-squares, a decomposition's linear solver, or some external source, four measures of goodness-of-fit respectively sum the *absolute* residuals (_Eigen_Misfit()) or the *squared* residuals (_Eigen_SSR()), quantify the percentage of total data variability in **B** accounted for by the model fit **X** (_Eigen_Rsquared()), or express the ratio of the residual norm over the total norm (_Eigen_RelativeError). These measures are also available in Col and Colwise variants, in order to handle solutions that contain multiple fits (that is, if observations matrix **B** has multiple columns).

LeastSquares

Function Reference

# _Eigen_LeastSquares

Solve a sytem of linear equations in the least-squares sense, using the normal equations

```
#include <Eigen4AutoIt.au3>
_Eigen_LeastSquares ( $matA, $matB[, $matW = 0[, $matX = 0[, $lambda =
0 ]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matW | **[optional]** matrix ID of the weighting matrix |
| $matX | **[optional]** matrix ID of the resulting model fit |
| $lambda | **[optional]** damping parameter |

## Return Value

Success: $matX

Failure: False, and sets the @error flag to non-zero.

## Remarks

See the Regression Tutorial for extensive instructions how to apply this function.

The main decompositions (LowerUpper, HouseolderQR, Choleski, and JacobiSVD) each provide their own linear solvers that provide alternative approaches to obtaining a linear model fit (See Eigen's documentation)

The number of rows of kernel matrix **A** has to match the number of rows of observations matrix **B**.

If a weighting matrix ($matW) is parsed, it has to be square with dimension rowsA, that is, its rows and columns should both match the number of rows of matrix **A**. To weight by the data covariance matrix **C** ("maximum likelihood", which accounts for interdependent effects), let $\mathbf{W} = \mathbf{A} \cdot \mathbf{C}^{-1} \cdot \mathbf{A}^{t}$, that is, multiply the kernel by the inverse covariance matrix by the kernel transposed, to obtain **W**.

If no existing results matrix (the model fit: $matX) of the correct dimensions is supplied (in this case, a matrix with the same number of rows as the number of columns of kernel matrix **A**, and the same number of columns as the number of columns of observations matrix **B**), a new results matrix will be created.

A non-zero (positive) $lambda calls the *damped* version of this function. In that case, $lambda should be tiny (but larger than the current precision level), because the larger $lambda is:

a) the lower the norm, that is, the smoother the solution,
b) the higher the misfit, and
c) the more $lambda will dominate the solution.

Damping thus provides regularisation of the solution at the expense of a poorer fit to the observations. To find an appropriate value, you can plot norm (obtained through MatrixSpecs) versus misfit for a range of $lambda values spanning several orders of magnitude (trade-off curve), and locate the point where the slope equals -1 (the "knee"). NB This is just one possible approach.

The LeastSquares function internally uses the robust LLDT version of the Choleski decomposition to obtain the inverse of the square product **($A^T.A$)**. A more robust decompositional alternative least-squares approach is provided by JacobiSVD; however, the latter is far slower and lacks optional weighting and damping.

If your data sufer from collinearity or other ill-conditioning, you can instead opt for a Partial Least-Squares analysis, by calling function _Eigen_PartialLeastSquares(). See that function's description for further details.

## Related

*Misfit(), SSR(), Rsquared(), PartialLeastSquares()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 ) ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

MSgBox ( 0, "Model residuals", "_Eigen_Misfit: " & _Eigen_Misfit ( $matA,
$matB, $matX ) & _
      @CR & "_Eigen_SSR    : " & _Eigen_SSR ( $matA, $matB, $matX ) & _
      @CR & "_Eigen_Rsquared: " & _Eigen_Rsquared ( $matA, $matB, $matX ) )
```

```
_Eigen_CleanUp()
```





# _Eigen_Misfit

Compute the sum of *absolute* residuals after obtaining a model fit

```
#include <Eigen4AutoIt.au3>
_Eigen_Misfit ( $matA, $matB, $matX )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |

## Return Value

| | |
|---|---|
| Success: | value (misfit) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The misfit (sum of absolute residuals) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated kernel ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The misfit represents one way to quantify their difference (SSR and Rsquared provide others).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

## Related

*LeastSquares*(), *SSR*(), *Rsquared*(), *RelativeError*(), *Misfit_Col*(), *Misfit_Colwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 )  ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

MSgBox ( 0, "Model residuals", "_Eigen_Misfit: " & _Eigen_Misfit ( $matA,
$matB, $matX ) & _
      @CR & "_Eigen_SSR: " & _Eigen_SSR ( $matA, $matB, $matX ) & _
      @CR & "_Eigen_RelativeError: " & _Eigen_RelativeError ( $matA, $matB,
$matX ) & _
      @CR & "_Eigen_Rsquared: " & _Eigen_Rsquared ( $matA, $matB, $matX ) )

_Eigen_CleanUp()
```





# _Eigen_Misfit_Col

Compute the sum of *absolute* residuals after obtaining a model fit, for a specified column of observations matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Misfit_Col ( $matA, $matB, $matX, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $colindex | column ID (base-0) of the column in matrix B to act upon |

## Return Value

| | |
|---|---|
| Success: | value (misfit) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The misfit (sum of absolute residuals) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated kernel ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The misfit represents one way to quantify their difference (SSR and Rsquared provide others).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

## Related

*LeastSquares*(), *SSR*(), *Rsquared*(), *RelativeError*(), *Misfit*(), *Misfit_Colwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 )  ; Colvector
```

```
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

$colindex = 0

MSgBox ( 0, "Model residuals", "_Eigen_Misfit_Col: " & _Eigen_Misfit_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_SSR_Col     : " & _Eigen_SSR_Col ( $matA, $matB, $matX,
$colindex ) & _
      @CR & "_Eigen_RelativeError_Col     : " & _Eigen_RelativeError_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_Rsquared_Col: " & _Eigen_Rsquared_Col ( $matA, $matB,
$matX, $colindex ) )

_Eigen_CleanUp()
```





# _Eigen_Misfit_Colwise

Compute the sum of *absolute* residuals after obtaining a model fit, per column of observations matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Misfit_Colwise ( $matA, $matB, $matX[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: value (misfit)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The misfit (sum of absolute residuals) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated kernel ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The misfit represents one way to quantify their difference (SSR and Rsquared provide others).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, a Rowvector with as many columns as the number of columns in matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*LeastSquares(), SSR(), Rsquared(), RelativeError(), Misfit_Col(), Misfit()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 2 )  ; two separate fits!
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 1 )
_Eigen_CwiseScalarOp_Col ( $matB, 1, "+", 1 )   ; points moved upward by 1
unit

_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0; intercept's factor 1

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
```

```
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "two model fits" )      ; row 0: slope; row 1:
intercept

; fitted line is also moved upward by 1 unit,
; so both sets of residuals (cols 0 and 1) are the same
$matM = _Eigen_Misfit_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matM, "Colwise Misfit" )

$matR = _Eigen_RelativeError_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Relative error" )

$matR = _Eigen_Rsquared_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Rsquared" )

$matS = _Eigen_SSR_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matS, "Colwise SSR" )

_Eigen_CleanUp()
```





# _Eigen_PartialLeastSquares

Solve a sytem of linear equations in the least-squares sense in eigenspace

```
#include <Eigen4AutoIt.au3>
_Eigen_PartialLeastSquares ( $matA, $matB[, $normalise =
True[,$addIntercept = True[, $lowerBound = 0.01[, $matR = 0[, $matS = 0[,
$matU = 0[, $matV = 0[, $matZ =0 ]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the inputs matrix (dimensions: [rowsA, colsA]) |
| $matB | matrix ID of the outputs matrix (dimensions: [rowsA, colsB]) |
| $normalise | **[optional]** **True**: normalise matrix A for SVD; **False**: don't |
| $addIntercept | **[optional]** **True**: add unity column to parameters prior to LSQ coefficients fitting; **False**: don't |
| $lowerbound | **[optional]** threshold to determine when eigenvalues are to be considered non-zero |
| $matR | **[optional]** matrix ID of the outputs' R-squared values [1,colsA] |
| $matS | **[optional]** matrix ID of the eigenvalues [1, colsA] (SVD output) |

| | |
|---|---|
| $matU | **[optional]** matrix ID of the eigenvectors [colsA, colsA] (SVD output) |
| $matV | **[optional]** matrix ID of the PCF results [colsA, colsB] |
| $matZ | **[optional]** matrix ID of the intercepts returned by the LSQ fit [1, colsB] |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

The main decompositions (LowerUpper, HouseolderQR, Choleski, and JacobiSVD) each provide their own linear solvers that provide alternative approaches to obtaining a linear model fit (See Eigen's documentation)

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on decompositions. See also Eigen's Linear Algebra Tutorial, and the Reference sections on modules SVD and Eigenvalues.

Inputs matrix **A** requires at least two columns and two rows (and preferably, rowsA >> colsA).
If $matA contains any column filled entirely with a single constant value, invalid results will be returned. An intercept column is automatically added as last column.

The number of rows of **inputs** matrix **A** has to match the number of rows of **outputs** matrix **B**. Note that matrix **B** is a required *input* for PartialLeastSquares.

The default $lowerbound value is greater than zero in order to filter out eigenvalues that are infinitesimally larger than zero, but would effectively still create a null space in the solution.

Best-fitting parameter coefficients plus intercept(s) in last row are returned in matrix **V**.

## Related

*Misfit(), SSR(), Rsquared(), LeastSquares()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp("double")

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]
```

```
$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 )  ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetRandom_Col ( $matA, 1 )   ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

_Eigen_PartialLeastSquares ( $matA, $matB )
$matV = _Eigen_GetActiveMatrix ( "V" )
$matZ = _Eigen_GetActiveMatrix ( "Z" )

_MatrixDisplay ( $matV, "model fit" )     ; coefficients
_MatrixDisplay ( $matV, "intercept" )     ; intercept(s)

_Eigen_CleanUp()
```





# _Eigen_PCA

## Perform a Principal Components Analysis (PCA)

```
#include <Eigen4AutoIt.au3>
_Eigen_PCA ( $matA[, $center = True[, $normalise = False[, $computeCovar =
True[, $returnPscores = False[, $returnQscores = True[, $matB = 0[, $matC
= 0[, $matD = 0[, $matM = 0[, $matP = 0[, $matQ = 0[, $matS = 0[, $matU =
0 ]]]]]]]]]]]] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the input matrix (dimensions: [rowsA, colsA]) |
| $center | **[optional]** **True**: center A in B, store colwise means in M; **False**: don't |
| $normalise | **[optional]** **True**: normalise tB; **False**: don't |
| $computeCovar | **[optional]** **True**: derive covariance matrix C from B; **False**: use pre-supplied data |
| $returnPscores | **[optional]** **True**: reproject A in P; **False**: don't |
| $returnQscores | **[optional]** **True**: reproject B in Q; **False**: don't |
| $matB | **[optional]** matrix ID of the centered matrix [rowsA, colsA] |
| $matC | **[optional]** matrix ID of the covariance matrix [colsA, colsA] |
| $matD | **[optional]** matrix ID of the percentage-variability-explained vector [1, colsA] |

| | |
|---|---|
| $matM | **[optional]** matrix ID of the colwise means vector [1, colsA] |
| $matP | **[optional]** matrix ID of the P scores matrix [colsA, rowsA] (A projected into the new space) |
| $matQ | **[optional]** matrix ID of the Q scores matrix [colsA, rowsA] (B projected into the new space) |
| $matS | **[optional]** matrix ID of the eigenvalues [1, colsA] (SVD output) |
| $matU | **[optional]** matrix ID of the eigenvectors [colsA, colsA] (SVD output) |

## Return Value

Success: True

Failure: False, and sets the @error flag to non-zero.

## Remarks

See the PCA Tutorial for extensive details on how this function works.

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Please consult Eigen's own Catalogue of decompositions for a general overview and links to extensive documentation on decompositions. See also Eigen's Linear Algebra Tutorial, and the Reference sections on modules SVD and Eigenvalues.

PCA, quantifies and maps the independent contributions to observed variability of the entire system,
This function accepts user-defined preset matrix activation(s) through _Eigen_SetActiveMatrix().

NB if $matA contains any column filled with a constant value, invalid results will be returned.

If $center = True (the default), the original data in $matA will be centered in $matB; the column means are then stored in $matM (so this matrix needs to be active; it is also required as input for the next step: _Eigen_PCA_ReDim()).
If $center = False, on the other hand, $matB is assumed to already contain the centered data, and $matM will not be created and/or filled.

If $normalise = True, $matB will contain the normalised centered data upon return. if $computeCovar is also True (the default), the analysis will use the **correlation** matrix (returned in $matC) instead of the **covariance** matrix (see below). Use the correlation matrix if the column variables measure properties with different distributions.
If $normalise = False, this step is skipped.

If $computeCovar = True (the default), covariance matrix $matC will be created and/or filled.
If $computeCovar = False, matrix $matC is expected to be pre-supplied (that is, filled!).

Eigenvalues are returned in Rowvector **S**; you can use _Eigen_Copy_Avector_ToBdiag() to turn this vector into a diagonal matrix.

Regarding the reprojections (a.k.a, the "scores"), either **P** or **Q**, none, or both can be returned. You can set either matrix to active (with _Eigen_SetActiveMatrix()) prior to calling _Eigen_PCA(), or alternatively, use booleans $returnPscores and $returnQscores to identify which result(s) should be returned. Note that **P** = (**U** * $\mathbf{B}^T$).rowwise() + (**U** * **M**), that is, **P** is simply **Q**, translated by (eigenvectors * column means), so returning both is redundant.

Matrices $matD (vector of eigenvalues as percentage of total variability explained, compare _Eigen_Rsquared()), $matS (eigenvalues proper, diagonal matrix), and $matU (eigenvectors, the PC "scores") are always created and/or filled.

## Related

*PCA_ReDim(), Decomp_JacobiSVD()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$matA = _Eigen_CreateMatrix_Random ( 6, 6 )     ; square
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "BCDMPQSU", True )     ; True = reset all flags first

_Eigen_PCA ( $matA )          ; the main call
If @error Then Exit

$matB = _Eigen_GetActiveMatrix ( "B" )    ; collect new IDs one at a time
$matC = _Eigen_GetActiveMatrix ( "C" )
$matD = _Eigen_GetActiveMatrix ( "D" )
$matM = _Eigen_GetActiveMatrix ( "M" )
$matP = _Eigen_GetActiveMatrix ( "P" )
$matQ = _Eigen_GetActiveMatrix ( "Q" )
$matS = _Eigen_GetActiveMatrix ( "S" )
$matU = _Eigen_GetActiveMatrix ( "U" )

_MatrixDisplay ( $matB, "PCA B matrix" )
_MatrixDisplay ( $matC, "PCA C matrix" )
_MatrixDisplay ( $matD, "PCA D matrix" )
_MatrixDisplay ( $matM, "PCA M matrix" )
_MatrixDisplay ( $matP, "PCA P matrix" )
_MatrixDisplay ( $matQ, "PCA Q matrix" )
_MatrixDisplay ( $matS, "PCA S matrix" )
_MatrixDisplay ( $matU, "PCA U matrix" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```



### PCA_ReDim

Function Reference

# _Eigen_PCA_ReDim

Perform dimensional compression after Principal Components Analysis (PCA)

```
#include <Eigen4AutoIt.au3>
_Eigen_PCA_ReDim ( $matU, $PCsToKeep = 0[, $matM = 0[, $matP = 0[, $matQ =
0[, $matR = 0 ]]]]] )
```

## Parameters

| | |
|---|---|
| $matU | matrix ID of the eigenvectors (dimensions: [colsA, colsA], see _Eigen_PCA()) |
| $PCsToKeep | the number of principal components to use in the back-projection (default: 0 = all) |
| $matM | **[optional]** matrix ID of the colwise means vector [1, colsA] |
| $matP | **[optional]** matrix ID of the P scores matrix [colsA, rowsA] |
| $matQ | **[optional]** matrix ID of the Q scores matrix [colsA, rowsA] |
| $matR | **[optional]** matrix ID of the results matrix [rowsA, colsA] |

## Return Value

| | |
|---|---|
| Success: | $matR |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

**First** call _Eigen_PCA() to generate all matrix inputs for this function!

See the PCA Tutorial for extensive details on how this function works.

Separate tables list the type and dimensions of each input and output for these (and related) procedures.

Integer value $PCsToKeep determines the dimensional reduction in the back-projection from the new orthonormal basis to the original reference frame in matrix **R**, where the value zero means keeping *all* principal components (no compression, recovering the original input), and integer values in the range [colsA to 1] indicate increasingly lossy compression, as additional PCs with the lowest-remaining eigenvalues are omitted.

Please provide either PCA outputs matrix **P** without matrices **Q** and **M**, or matrices **Q** and **M** without matrix **P**.

If no existing results matrix **R** of the correct dimensions is supplied (in this case, a matrix with the same dimensions as matrix **A**, the original input matrix of _Eigen_PCA()), a new results matrix will be created.

## Related

*PCA()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()
```

```
$matA = _Eigen_CreateMatrix_Random ( 8, 6 )     ; 8 observations in 6
dimensions (variables) each
_MatrixDisplay ( $matA, "matrix A" )

_Eigen_SetActiveMatrix ( "PU", True )     ; True = reset all flags first

_Eigen_PCA ( $matA )    ; first call PCA itself, then ReDim
If @error Then Exit

$matP = _Eigen_GetActiveMatrix ( "P" )
_MatrixDisplay ( $matP, "reprojected data" )

$matU = _Eigen_GetActiveMatrix ( "U" )
_MatrixDisplay ( $matU, "eigenvectors" )

$PCs_toKeep = 2         ; dimensional reduction (from 6 to 2)

$matR = _Eigen_PCA_ReDim ( $matU, $PCs_toKeep, 0, $matP )   ; 0 = $matM is not
used here
_MatrixDisplay ( $matR, "A compressed" )

_Eigen_ResetActiveMatrix()

_Eigen_CleanUp()
```



### RelativeError

Function Reference

# _Eigen_RelativeError

Compute the ratio of a model fit's residual norm over the observational norm

```
#include <Eigen4AutoIt.au3>
_Eigen_RelativeError ( $matA, $matB, $matX )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |

## Return Value

Success: value (misfit)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The relative error (((**A.X**)-**B**).norm() / **B**.norm(), where norm = Euclidean $L^2$ norm) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated kernel ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The relative error represents one way to quantify their difference (the Misfit, SSR, and Rsquared provide others).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

## Related

*LeastSquares*(), *SSR*(), *Rsquared*(), *RelativeError_Col*(), *RelativeError_Colwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 )  ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

MSgBox ( 0, "Model residuals", "_Eigen_Misfit: " & _Eigen_Misfit ( $matA,
$matB, $matX ) & _
      @CR & "_Eigen_SSR: " & _Eigen_SSR ( $matA, $matB, $matX ) & _
      @CR & "_Eigen_RelativeError: " & _Eigen_RelativeError ( $matA, $matB,
$matX ) & _
      @CR & "_Eigen_Rsquared: " & _Eigen_Rsquared ( $matA, $matB, $matX ) )
```

```
_Eigen_CleanUp()
```



RelativeError_Col

Function Reference

# _Eigen_RelativeError_Col

Compute the ratio of a model fit's residual norm over the observational norm, for a specified column of observations matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_RelativeError_Col ( $matA, $matB, $matX, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $colindex | column ID (base-0) of the column in matrix B to act upon |

## Return Value

| | |
|---|---|
| Success: | value (misfit) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The relative error (((**A.X**)-**B**).norm() / **B**.norm(), where norm = Euclidean $L^2$ norm) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated kernel ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The relative error represents one way to quantify their difference (the Misfit, SSR, and Rsquared provide others).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

## Related

*LeastSquares*(), *SSR*(), *Rsquared*(), *RelativeError*(), *RelativeError_Colwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 )  ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

$colindex = 0

MSgBox ( 0, "Model residuals", "_Eigen_Misfit_Col: " & _Eigen_Misfit_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_SSR_Col     : " & _Eigen_SSR_Col ( $matA, $matB, $matX,
$colindex ) & _
      @CR & "_Eigen_RelativeError_Col     : " & _Eigen_RelativeError_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_Rsquared_Col: " & _Eigen_Rsquared_Col ( $matA, $matB,
$matX, $colindex ) )

_Eigen_CleanUp()
```



### RelativeError_Colwise

Function Reference

# _Eigen_RelativeError_Colwise

Compute the ratio of a model fit's residual norm over the observational norm, per column of observations matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_RelativeError_Colwise ( $matA, $matB, $matX[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

| | |
|---|---|
| Success: | value (misfit) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The relative error (((**A.X**)-**B**).norm() / **B**.norm(), where norm = Euclidean $L^2$ norm) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated kernel ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The relative error represents one way to quantify their difference (the Misfit, SSR, and Rsquared provide others).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, a Rowvector with as many columns as the number of columns in matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*LeastSquares*(), *SSR*(), *Rsquared*(), *RelativeError*(), *RelativeError_Col*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 2 ) ; two separate fits!
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 1 )
_Eigen_CwiseScalarOp_Col ( $matB, 1, "+", 1 )   ; points moved upward by 1
unit

_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0; intercept's factor 1

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "two model fits" )      ; row 0: slope; row 1:
intercept

; fitted line is also moved upward by 1 unit,
; so both sets of residuals (cols 0 and 1) are the same
$matM = _Eigen_Misfit_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matM, "Colwise Misfit" )

$matR = _Eigen_RelativeError_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Relative error" )

$matR = _Eigen_Rsquared_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Rsquared" )

$matS = _Eigen_SSR_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matS, "Colwise SSR" )

_Eigen_CleanUp()
```



## RsqAdjusted

Function Reference

# _Eigen_RsqAdjusted

Compute the percentage of total variability explained by a model fit, adjusted for overfitting

```
#include <Eigen4AutoIt.au3>
_Eigen_RsqAdjusted ( $matA, $matB, $matX )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |

## Return Value

Success: value (adjusted R-squared)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The R-squared statistic (a.k.a. "coefficient of determination") provides a measure of how well a model ($matX) fits the observations ($matB), given the associated equations of condition ($matA). Most decompositions, as well as _Eigen_LeastSquares(), have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. R-squared expresses the percentage of total data variability explained by the predictions; other measures of this difference are Misfit (sum of absolute residuals) and SSR (sum of squared residuals)

The **adjusted** version of R-squared takes any potential overfitting of superfluous parameters ("kitchen sink regression") into account, by returning instead: $1 - ( 1 - R^2 ) * [ ( n - 1 ) / ( n - p - 1 ) ]$, with $n$ the sample size and $p$ the number of fitted parameters. Thus the adjusted $R^2$ is always less than the original $R^2$, and a sizeable decrease would indicate overfitting (some parameters explain no additional variability in the observations).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

## Related

*LeastSquares(), Misfit(), SSR(), RelativeError(), Rsquared_Col(), Rsquared_Colwise()*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 ) ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

MSgBox ( 0, "Model residuals", "_Eigen_Misfit: " & _Eigen_Misfit ( $matA,
$matB, $matX ) & _
      @CR & "_Eigen_SSR: " & _Eigen_SSR ( $matA, $matB, $matX ) & _
      @CR & "_Eigen_RelativeError: " & _Eigen_RelativeError ( $matA, $matB,
$matX ) & _
      @CR & "_Eigen_Rsquared: " & _Eigen_Rsquared ( $matA, $matB, $matX ) )

_Eigen_CleanUp()
```



### Rsquared

Function Reference

# _Eigen_Rsquared

Compute the percentage of total variability explained by a model fit

```
#include <Eigen4AutoIt.au3>
_Eigen_Rsquared ( $matA, $matB, $matX )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |

## Return Value

Success: value (R-squared)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The R-squared statistic (a.k.a. "coefficient of determination") provides a measure of how well a model ($matX) fits the observations ($matB), given the associated equations of condition ($matA). Most decompositions, as well as _Eigen_LeastSquares(), have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. R-squared expresses the percentage of total data variability explained by the predictions; other measures of this difference are Misfit (sum of absolute residuals) and SSR (sum of squared residuals)

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

## Related

*LeastSquares(), Misfit(), SSR(), RelativeError(), Rsquared_Col(), Rsquared_Colwise(),RsqAdjusted*

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 )  ; Colvector
```

```
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

MSgBox ( 0, "Model residuals", "_Eigen_Misfit: " & _Eigen_Misfit ( $matA,
$matB, $matX ) & _
      @CR & "_Eigen_SSR: " & _Eigen_SSR ( $matA, $matB, $matX ) & _
      @CR & "_Eigen_RelativeError: " & _Eigen_RelativeError ( $matA, $matB,
$matX ) & _
      @CR & "_Eigen_Rsquared: " & _Eigen_Rsquared ( $matA, $matB, $matX ) )

_Eigen_CleanUp()
```





# _Eigen_Rsquared_Col

Compute the percentage of total variability explained by a model fit, for a specified column of observations matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Rsquared_Col ( $matA, $matB, $matX, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $colindex | column ID (base-0) of the column in matrix B to act upon |

## Return Value

Success: value (R-squared)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The R-squared statistic (a.k.a. "coefficient of determination") provides a measure of how well a model ($matX) fits the observations ($matB), given the associated equations of condition ($matA). Most decompositions, as well as _Eigen_LeastSquares(), have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. R-squared expresses the percentage of total data variability explained by the predictions; other measures of this difference are Misfit (sum of absolute residuals) and SSR (sum of squared residuals)

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

## Related

*LeastSquares*(), *Misfit*(), *SSR*(), *RelativeError*(), *Rsquared*(), *Rsquared_Colwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 ) ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

$colindex = 0

MSgBox ( 0, "Model residuals", "_Eigen_Misfit_Col: " & _Eigen_Misfit_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_SSR_Col    : " & _Eigen_SSR_Col ( $matA, $matB, $matX,
```

```
$colindex ) & _
      @CR & "_Eigen_RelativeError_Col     : " & _Eigen_RelativeError_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_Rsquared_Col: " & _Eigen_Rsquared_Col ( $matA, $matB,
$matX, $colindex ) )

_Eigen_CleanUp()
```





# _Eigen_Rsquared_Colwise

Compute the percentage of total variability explained by a model fit, per column of observations matrix B

```
#include <Eigen4AutoIt.au3>
_Eigen_Rsquared_Colwise ( $matA, $matB, $matX[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

Success: value (R-squared)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The R-squared statistic (a.k.a. "coefficient of determination") provides a measure of how well a model ($matX) fits the observations ($matB), given the associated equations of condition ($matA). Most decompositions, as well as _Eigen_LeastSquares(), have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. R-squared expresses the percentage of total data variability explained by the predictions; other measures of this difference are Misfit (sum of absolute residuals) and SSR (sum of squared residuals)

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, a Rowvector with as many columns as the number of columns in matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*LeastSquares*(), *Misfit*(), *SSR*(), *RelativeError*(), *Rsquared_Col*(), *Rsquared*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 2 ) ; two separate fits!
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 1 )
_Eigen_CwiseScalarOp_Col ( $matB, 1, "+", 1 )   ; points moved upward by 1
unit

_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0; intercept's factor 1

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "two model fits" )      ; row 0: slope; row 1:
intercept

; fitted line is also moved upward by 1 unit,
; so both sets of residuals (cols 0 and 1) are the same
$matM = _Eigen_Misfit_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matM, "Colwise Misfit" )

$matR = _Eigen_RelativeError_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Relative error" )
```

```
$matR = _Eigen_Rsquared_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Rsquared" )

$matS = _Eigen_SSR_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matS, "Colwise SSR" )

_Eigen_CleanUp()
```



SSR

Function Reference

# _Eigen_SSR

Compute the sum of *squared* residuals after obtaining a model fit

```
#include <Eigen4AutoIt.au3>
_Eigen_SSR ( $matA, $matB, $matX )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |

## Return Value

Success: value (SSR)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The SSR (sum of squared residuals) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated equations of condition ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The SSR represents one way to quantify their difference (others are Misfit and Rsquared).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

## Related

*LeastSquares*(), *Misfit*(), *Rsquared*(), *RelativeError*(), *SSR_Col*(), *SSR_Colwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 ) ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

MSgBox ( 0, "Model residuals", "_Eigen_Misfit: " & _Eigen_Misfit ( $matA,
$matB, $matX ) & _
      @CR & "_Eigen_SSR: " & _Eigen_SSR ( $matA, $matB, $matX ) & _
      @CR & "_Eigen_RelativeError: " & _Eigen_RelativeError ( $matA, $matB,
$matX ) & _
      @CR & "_Eigen_Rsquared: " & _Eigen_Rsquared ( $matA, $matB, $matX ) )

_Eigen_CleanUp()
```



### SSR_Col

Function Reference

# _Eigen_SSR_Col

Compute the sum of *squared* residuals after obtaining a model fit, for a specified column of observations matrix B

```
#include <Eigen4AutoIt.au3>
```

```
_Eigen_SSR_Col ( $matA, $matB, $matX, $colindex )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $colindex | column ID (base-0) of the column in matrix B to act upon |

## Return Value

Success: value (SSR)

Failure: False, and sets the @error flag to non-zero.

## Remarks

The SSR (sum of squared residuals) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated equations of condition ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The SSR represents one way to quantify their difference (others are Misfit and Rsquared).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

## Related

*LeastSquares*(), *Misfit*(), *Rsquared*(), *RelativeError*(), *SSR*(), *SSR_Colwise*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
```

```
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 1 )  ; Colvector
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "model fit" )     ; row 0: slope; row 1: intercept

$colindex = 0

MSgBox ( 0, "Model residuals", "_Eigen_Misfit_Col: " & _Eigen_Misfit_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_SSR_Col     : " & _Eigen_SSR_Col ( $matA, $matB, $matX,
$colindex ) & _
      @CR & "_Eigen_RelativeError_Col     : " & _Eigen_RelativeError_Col
( $matA, $matB, $matX, $colindex ) & _
      @CR & "_Eigen_Rsquared_Col: " & _Eigen_Rsquared_Col ( $matA, $matB,
$matX, $colindex ) )

_Eigen_CleanUp()
```





# _Eigen_SSR_Colwise

Compute the sum of *squared* residuals after obtaining a model fit

```
#include <Eigen4AutoIt.au3>
_Eigen_SSR_Colwise ( $matA, $matB, $matX[, $matR = 0 ] )
```

## Parameters

| | |
|---|---|
| $matA | matrix ID of the kernel matrix |
| $matB | matrix ID of the observations matrix |
| $matX | matrix ID of the model fit |
| $matR | **[optional]** matrix ID of the results matrix |

## Return Value

| | |
|---|---|
| Success: | value (SSR) |
| Failure: | False, and sets the @error flag to non-zero. |

## Remarks

The SSR (sum of squared residuals) provides a measure of how well a model ($matX) fits the observations ($matB), given the associated equations of condition ($matA). Most decompositions, as well as **_Eigen_LeastSquares()**, have the ability to solve a set of linear equations of the form **A . X = B**. Residuals can then be computed by subtracting the model predictions for any **A** from the actual observations. The SSR represents one way to quantify their difference (others are Misfit and Rsquared).

The number of rows in matrix **A** has to match the number of rows in matrix **B**.
The number of rows in matrix **X** has to match the number of columns in matrix **A**.
The number of columns in matrix **X** has to match the number of columns in matrix **B**.

This specific version of this measure is useful when observations matrix **B** contains multiple colums, causing the linear solver to produce as many fits in the columns of model matrix **X**, which can be targeted individually (variant: _Col) or as a set (variant: _Colwise).

If results matrix **R** is pre-supplied, it has to have the correct dimensions (in this case, a Rowvector with as many columns as the number of columns in matrix **B**) If results matrix **R** is not pre-supplied, it will be created.

## Related

*LeastSquares*(), *Misfit*(), *Rsquared*(), *RelativeError*(), *SSR_Col*(), *SSR*()

## Example

```
#include "Eigen4AutoIt.au3"

_Eigen_StartUp()

$rows = 10
Local $arrayA [ $rows ][ 2 ] = [ _
      [ 2.5, 2.4 ], _    ; X,Y data (observation pairs)
      [ 0.5, 0.7 ], _    ; X: independent (condition)
      [ 2.2, 2.9 ], _    ; Y: dependent (response)
      [ 1.9, 2.2 ], _
      [ 3.1, 3.0 ], _
      [ 2.3, 2.7 ], _
      [ 2.0, 1.6 ], _
      [ 1.0, 1.1 ], _
      [ 1.5, 1.6 ], _
      [ 1.1, 0.9 ] ]

$matA = _Eigen_CreateMatrix_FromArray ( $arrayA )

$matB = _Eigen_CreateMatrix ( $rows, 2 )  ; two separate fits!
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 0 )
_Eigen_Copy_Acol_ToBcol ( $matA, $matB, 1, 1 )
```

```
_Eigen_CwiseScalarOp_Col ( $matB, 1, "+", 1 )   ; points moved upward by 1
unit

_Eigen_SetOnes_Col ( $matA, 1 )     ; colindex = base-0; intercept's factor 1

_MatrixDisplay ( $matA, "kernel A" )      ; col 0: X values; col 1: intercept
factor (1)
_MatrixDisplay ( $matB, "matrix B (obs)" )      ; response that the model
should fit

$matX = _Eigen_LeastSquares ( $matA, $matB )
_MatrixDisplay ( $matX, "two model fits" )      ; row 0: slope; row 1:
intercept

; fitted line is also moved upward by 1 unit,
; so both sets of residuals (cols 0 and 1) are the same
$matM = _Eigen_Misfit_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matM, "Colwise Misfit" )

$matR = _Eigen_RelativeError_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Relative error" )

$matR = _Eigen_Rsquared_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matR, "Colwise Rsquared" )

$matS = _Eigen_SSR_Colwise ( $matA, $matB, $matX )
_MatrixDisplay ( $matS, "Colwise SSR" )

_Eigen_CleanUp()
```



## Appendix

# Appendix Reference

- Eigen4AutoIt Globals
- Eigen4AutoIt Limitations
- Eigen4AutoIt Error Codes



### Eigen4AutoIt Globals

# Eigen4AutoIt Globals

The work environment comprises a number of internal global variables, whose content should only ever be altered by calling the appropriate functions (with one exception: **$EIGEN_DLLPATH**, which you set once, after installing the software):

| Global Variable | Description |
|---|---|
| **$EIGEN_ABORTEXITS** | <boolean>; True: fatal E4A error Exits, False: fatal E4A error Returns |

| | |
|---|---|
| **$EIGEN_ACTIVE_MATRICES** | <int>; bit flags for each matrix to create / act on / return (0-25 = **A-Z**, 26=Specs (scalars), remainder reserved) |
| **$EIGEN_BYTESPERELEMENT** | <int> real float, int: 4; real double: 8; complex float: 2 x 4; complex double: 2 x 8 |
| **$EIGEN_COMPLEX** | <boolean>; True: matrix type is complex, False: matrix type is real |
| **$EIGEN_COMPLEXREALIMAG** | <string>: internal dll call suffix to select _Real/_Imag complex sub-functions |
| **$EIGEN_DEBUGMODE** | <boolean>; True: Debug mode active, False: Computing Mode active |
| **$EIGEN_DECIMALS** | <int>; affects precision and rounding; related to **$EIGEN_MATRIXTYPE** |
| **$EIGEN_DENSE_DLLHANDLE** | <string>; dll handle for **Eigen4AutoIt**'s dense matrix library (either the slow "Debug" or the fast "Computing" version) |
| **$EIGEN_DLLPATH** | <string>; full path of the directory where the ***EigenDense*** dlls are stored (includes drive letter; no trailing backslash); **you define this one *once*, upon installation.** |
| **$EIGEN_DLLSUFFIX** | <string>;  matrixtype dll call's function name suffix |
| **$EIGEN_FLAGS** | <int>; bit flags for operational settings in specific dll calls (see below) |
| **$EIGEN_LASTERROR** | <int>; most recent E4A error code |
| **$EIGEN_MATRIXTYPE** | <string>; default Eigen matrix type (afects storage size and precision) |
| **$EIGEN_MAXINTEGERWIDTH** | <int>; for file conversion with fixed-width fields |
| **$EIGEN_TYPELABEL** | <string>; numeric type for variables parsed to dll; related to **$EIGEN_MATRIXTYPE** |
| **$EIGEN_VARTYPE** | <int>; matrix type as stored in global **$MatrixList** and file headers; see below for contents |
| **$MATRIXLIST** | array repository for all defined matrix structs plus their associated dimensions, variable type, memory pointer, and the function that initially created it |

In addition, the work environment relies upon a number of global constants, partly related to the matrix types listed here; the operational $FLAGS set the designated bit in global variable **$EIGEN_FLAGS**.

| Global Constant | Integer value |
|---|---|
| **$EIGEN_MATRIXFILEMARKER** | **202,331,218**; the first 4 bytes of any matrix file saved in the **Eigen4AutoIt** environment; this marker can be used as BOM to determine byte-order in other environments. |
| **$VARTYPE_complexdouble** | **6**; currently supported only for type conversion purposes (2 x 8 bytes per cell) |
| **$VARTYPE_complexfloat** | **4**; currently supported only for type conversion purposes (2 x 4 bytes per cell) |
| **$VARTYPE_int** | **1**; supported only for type conversion purposes, as scientific computing is rarely restricted to integer values |

| | |
|---|---|
| **$VARTYPE_realdouble** | **2**; double precision; stores 8 bytes per matrix element |
| **$VARTYPE_realfloat** | **0**; single precision (the default); stores 4 bytes per matrix element |
| **$FLAGS_ComputeThin** | **0**; generic decomposition operational setting |
| **$FLAGS_ColPivoting** | **1**; generic decomposition operational setting |
| **$FLAGS_FullPivoting** | **2**; generic decomposition operational setting |
| **$FLAGS_CHOL_Robust** | **3**; Choleski decomposition operational setting |
| **$FLAGS_SVD_QRPreconditioner** | **4**; JacobiSVD decomposition operational setting |
| **$FLAGS_SCHUR_ComputeFromHessenberg** | **5**; RealSchur helper decomposition operational setting |
| **$FLAGS_PCA_Normalise** | **6**; PCA/PCF operational setting |
| **$FLAGS_PCA_ComputeCoveriance** | **7**; PCA operational setting |
| **$FLAGS_ES_EigenvaluesAsVector** | **8**; JacobiSVD & Eigensolvers operational setting |
| **$FLAGS_ES_ABx_lambdaX** | **9**; Generalized Self-Adjoint Eigensolver operational setting |
| **$FLAGS_ES_BAx_lambdaX** | **10**; Generalized Self-Adjoint Eigensolver operational setting |
| **$FLAGS_PLS_Normalise** | **11**; Partial Least Squares operational setting |
| **$FLAGS_PLS_AddIntercept** | **12**; Partial Least Squares operational setting |

Finally, as of version 2.2, **Eigen4AutoIt** supports about one thousand global constants from mathematics and phyics, stored in include ***E4Aconstants.au3***. See the description of functions _Eigen_Show_Constants_*() in subsection Work Environment of the Function Reference to learn more.



### Eigen4AutoIt Limitations

# Eigen4AutoIt Limitations

| Eigen limitations | Value | Description |
|---|---|---|
| automatic matrix type conversion | none | not supported |
| vector cross product | 3 elements | other sizes not (yet) supported |
| HouseholderQR linear solver | vectors only | matrices not (yet) supported |
| EigenSolver_Generalized | eigenvectors | not supported |

| AutoIt3 limitations | Value | Description |
|---|---|---|
| MAX_BINARYSIZE | 2,147,483,647 | maximum bytes of binary data |
| MAX_LINESIZE | 4095 | maximum size for a line of script |

| | | |
|---|---|---|
| VAR_SUBSCRIPT_ELEMENTS | 16,777,216 | maximum number of elements for an array |
| VAR_SUBSCRIPT_MAX | 64 | maximum number of subscripts for an array |
| <implicit> | 65,525 | maximum number of rows displayed by _MatrixDisplay |

(copied in part from the AutoIt v3.3.12 documentation)

| Eigen4AutoIt limitations | Functions | Description |
|---|---|---|
| Geometry module | none | currently not supported |
| Sparse Matrix module | none | ain't gonna happen! |
| JacobiRotation | none | currently not supported |
| HouseholderSequence | none | currently not supported |
| computeInverseWithCheck | none | superfluous, weak algorithm |
| computeInverseAndDetWithCheck | none | superfluous, weak algorithm |
| Choleski UpLo option | Choleski | superfluous; Lower = default, always |
| Choleski rank-update with sigma and vectorW | Choleski | not applicable in this environment |
| ad hoc expression templates | none | not applicable in this environment |
| temporary objects | none | not applicable in this environment |



### Eigen4AutoIt Error Codes

# Eigen4AutoIt Error Codes

| Error code | Explanation |
|---|---|
| **-1** | generic / internal / unspecified error |
| **-2** | wrong / mismatched dimension(s) / shape / coordinate(s) |
| **-3** | matrix pointer / index / expected-presence issue |
| **-4** | wrong matrix type (int, float, double, complex float, complex double) |
| **-5** | file I/O error |
| **-6** | violated limitation associated with function |
| **-7** | failed dll call / failed Eigen call |
| **-8** | illegal self-reference / aliasing issue |
| **-9** | invalid parameter (specID, operator, matrix letter, PCsToKeep...) |

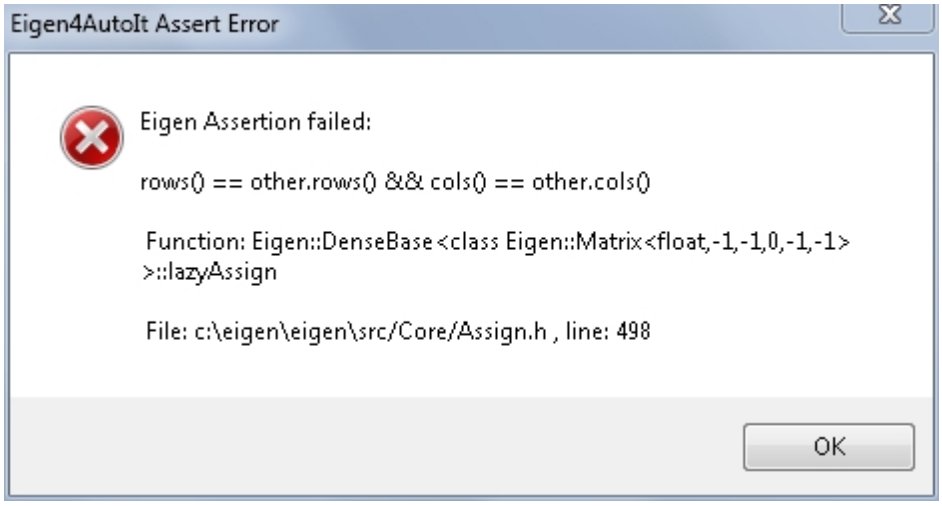
Internally, **Eigen4Autoit** employs call stack tracing to keep track of potentially deeply-nested procedure calls. This means that whenever @error is set to non-zero by an **Eigen4AutoIt** function, a dedicated

internal error handler will produce an informative error message on screen that will always include *the complete function calling tree* that gave rise to the error. Note that this handler is part of the **AutoIt** side of the **Eigen** interface, and cannot (easily) be switched off.

Some common scenarios for errors:

- If you keep getting @error = **-7** with the message: "Dllcall failed with error: **1** (= "unable to use the DLL file", see DllCall in AutoIt Help), then you probably forgot to call _Eigen_Startup() before calling other _Eigen functions.
- Do **NOT** mix matrices of different matrix type in a single _Eigen call; Eigen itself does not support automatic type conversion. Instead, explicitly convert the matrix type either on file or in memory, to match the current matrix type of your work environment. Conversions in memory always create a new matrix as output; calling the file conversion functions with a single filename will rewrite the file itself (which may cause loss of information if the new type uses less storage), whereas calling with two filenames will interpret these as source and destination (the latter will be overwritten if pre-existing).
- Do not confuse a *Row*vector (1 x N; multiple columns) with a *Col*vector (N x 1; multiple rows).
- All indices are **base-0**.
- When importing array data from external sources into a matrix, please check whether any array element *could* contain a scientific notation string (e.g., "0.123456-e012"). If so, you'll want to set the flag $convertScientificNotation = **True** (default: **False**, because conversion takes extra time), otherwise these strings (like any other strings) will be stored as zeroes.
- Functions with multiple input matrices often impose constraints on one or both of their dimensions.
- If you supply an existing output container, please ensure that the result will *exactly* fit that container. A container that is too big will fail just as hard as one that is too small.
- If you define a block inside a matrix, no block edge may extend beyond the matrix bounds.
- Parsed integer parameters that designate a specific sub-function (e.g., $specID) not only have a limited range, but may also apply *only* to particular input types (full matrix versus matrix part), or be otherwise restricted in use (e.g., MatrixSpecs).
- **A matrix inverse may not exist!** Always check that an inversion yields a valid result. If inverting a square matrix, you can check beforehand whether its determinant is zero (see MatrixSpecs), in which case no inverse exists. If you have to produce an inverse no matter what, you may find JacobiSVD's *pseudo-inverse* (Penrose_Moore) and function _Eigen_PseudoInverse of interest.
- Most decompositions have important limitations on the properties of acceptable input matrices. Disregarding these may cause the call to A) fail, or B) return total garbage. You have been warned.

**Eigen** also has its own dedicated error-checking asserts (embedded into the ***EigenDense_Debug.dll***) that only partially overlap with those in **Eigen4AutoIt**. By default, these (time-consuming) checks are switched on, and will produce their own "failed assert" message on screen when triggered. See the section Work Environment on how to switch this feature on and off, and the Basics Tutorial on how to interpret these error messages.




Lastly, **Eigen**'s decompositions and EigenSolvers all generate a (zero or positive) status code, which is captured and incorporated into **Eigen4AutoIt** error messages where appropriate. This affects all decomposition-related calls, including eigensolvers and **_Eigen_PCA()**. Note that **Eigen**'s status code zero means success, whereas **Eigen4AutoIt**'s regular dll function status code 0 (False) means failure (and 1 = True = success). Status code 4 was added for this environment. The current status codes are:

| Status code | Explanation |
|---|---|
| **0** | Succes. |
| **1** | Numerical issue: the provided data did not satisfy the prerequisites. |
| **2** | No convergence: iterative procedure did not converge. |
| **3** | Invalid input: the inputs are invalid, or the algorithm has been improperly called. |
| **4** | Complex output: unable to store imaginary part of complex output(s) in containers for reals. |